JAVA

Bölüm 1: Java'ca Konuşmaya Başlamak

İstanbulda soğuk bir kış günüydü, odamın camından lapa lapa yağan kar yağışını seyrediyordum. Her yer bembeyaz olmuştu. Evden çıkmak bile bu tipide delilik olurdu. Ama gelişmelerin beni evden çıkartacağını bilemezdim. Elimdeki bir fincan kahveyi yudumlarken, aklıma bugüne kadar uğraştığım programlama dilleri geldi. Üniversitede Delphi, sonra C++ ve son olarakta C#. Aralarda ufak çaplı olsada, Visual Basic ilede ilgilenmiştim. Son kahvemden bir yudum daha aldım ve aklıma

ilginç bir fikir geliverdi. Neden Java ile hiç uğraşmamıştım. C dilinin kurallarına ve yazım tarzına çok benzeyen gerçek anlamda nesne yönelimli, performansı yüksek, platform bağımısız olan bir dil.

Aslında 90lı yılların başında HotJava ile ilk kez tanıtıldığını bildiğim bu dil zaman içerisinde hiçde yadsınamayacak bir ilerleme ve gelişme göstermişti. Böylesine değerli bir dili görmezden gelmek, bir programcı için iyi olmaz diye düşündüm. O halde karar verilmişti. Sadece bir kahve molasında, Java öğrenmeye karar vermiştim. Evet çok güzel. Ama nerden ve nasıl başlamalıyım.

Önceki deneyimlerim, bir bilgisayar programlama dilinin nasıl işlediğini, nasıl çalıştığını anlamak ve amatör düzeyde uygulamalar geliştirmek için, merakın başladığı andan itibaren, en az bir ay süreli sıkı bir çalışma gerektirdiği yönündeydi. Nitekim konstantrasyon kaybolmadan, mümkün olduğu kadar çok kaynaktan eş zamanlı olarak bir çalışma yaparak başarıya ulaşmaştım hep. Öyleyse izleyeceğim yol belliydi. Şimdi bana bolca kaynak lazım. Internet üzerinde Java ile ilgili sayısız kaynak var. Hatta, Kazaa gibi programlar ile, java üzerine yazılmış pek çok elektronik kitaba ulaşmakta mümkün.

Ancak, doğruyu söylemek gerekirse, ekran başında kalıp saatlerce bir dökümanı okumak bence sağlık açısından pek iyi bir çalışma sistemi değil. Herşeyden önce çalışma alanının bize huzur veren rahat bir ortam olması gerekiyor. Geniş bir ortam olabilir. Huzuru ise ben çoğunlukla ece çok geç saatlerde bütün mahalle yattığında bulabiliyorum. Alanı geniş olan bir masa ve bu konuda yazılmış kaynaklara ait dökümanlar. Bu düşünceler ile yola çıkarak önce Java ile ilgili en güncel kitapların neler olduğunu internet vasıtasıyla araştırdım. Bu gün itibariyle aşağıdaki tabloda yer alan kitaplar Java üzerine yazılmış en güncel Türkçe kaynaklar.

Sırada bu kaynakları edinmek var. Bunu ertesi günde gerçekleştirebilirim. Çünkü dışarıda bir tipi var. Ama içimdeki öğrenme ve merak hisleri beni bu konuda harekete geçiriyor. O gün o tipide dışarı çıktım, Kadıköy�e gittim ve buradaki kitabevlerini, sahafları gezerek yukarıdaki kitapları aradım.

Zaten bu kitaplar oldukça etkili ve iyi basımlar olduğu için her kitabevinde bulmak mümkün. Hepsi biribirinden değerli yazarlar ve yayınevlerince çıkartılmış. Kitapların hepsini temin ettim. Oldukça fazla tuttu tabi. Ama öğrenmek için ilk başta verilecek bu tutar, öğrendiklerimiz ile yapabileceklerimiz düşünüldüğünde göze alınacak bir miktar gibi geldi. Eve döndüğümde bir buz kalıbından farksızdım. Sırt çantam Java dilinin ağırlığını taşıyan kitaplar ile bir kaç kart daha artmış bir halde neredeyse boyumu kısaltmıştı. Büyük bir heyecanla kitapları çalışma masamın üzerine serdim. Nasıl bir hızla ve hevesle başladım bilmiyorum ama hava karardığında, bu dil ile ilgili ilk izlenimlerimi edinmiş ve saygın bir görüşe sahip olmuştum.

Java daha önceden bahsettiğim gibi, 90lı yıllarda doğmuş bir dil idi. Üretici firma Sun. Java ile yazılan programların ilk ve en ayır edici özellikleri, platform bağımsız olmaları. Bunu nasıl sağladıklarını araştırdığımda, Virtual Java Machine (Sanal Java Makinesi) kavramı ile karşılaştım. Bu aslında günümüz işletim sistemlerinin tümünde bulunuyor. Hatta kullanıdığımız web tarayıcılarının sistemimizde olması bile yeterli. Bu noktada kafam karışmıştı. Bu ara programın amacı neydi acaba? Çok geçmeden cevabı buldum. Biz java dili ile bir program yazdığımızda bunu Java Derleycisi ile derliyoruz. Derlenmiş olan bu dosyalar bytecode adı verilen bir hale geliyor. JVM ise, derlenmiş bir java uygulaması başlatıldığında, bu uygulamayı sisteme uygun bir halde yorumluyor ve çalıştırıyor. Dolayısıyla JVM ara programının yüklü olduğu her sistemde bu kodları değiştirmeden çalıştırabilme imkanına sahibiz. Sanıyorumki bugünlerde mobil telefonlarda java ile yazılmış oyunların çokluğunun sebebi bu olsa gerek. İlk öğrendiğim bu kavram aslında bu programlama dilinin gücünü ve her programcının az da olsa gilmesi gerektiğini gösterdi bana. O halde koşmak lazım. Hedefimiz büyük.

Elbette kuru kuruya bunları yazmak kolay. Ama basit bir uygulamada geliştirmek ve bir yerden başlamak gerekli. Öncelikle benim Java programlarını yazmam için gereken bir takım şeyler var bundan eminim. Çünkü bu her zaman böyle olmuştur. İhtiyacım olan ilk ve hatta tek şeyin, JSDK (Java Software Developer Kit-Java Yazılım Geliştirme Kiti) olduğunu öğreniyorum. Bunu kitaplar ile birlikte gelen cd�lerden temin

edebiliriz. Ama bence en güncel olan sürümü takip etmek ve kullanmak her zaman daha iyidir. Küçük bir araştırma sonucu aşağıdaki adrese giriyor ve JSDK�ların yayınlanmış tüm sürümlerine ulaşıyorum.

Ben burada Offline Kurulum seçeneğini seçtim. J2SE için yaklaşık olarak 15 megabyte�lık bir dosya indirdi. Artık indirilen bu dosyayı sistemime kurabilirim. Kurulum adımları son derece kolay. Yazılım kitinin kurulması ile birlikte artık sistemim, java uygulamalarını geliştirmem için uygun bir yapıya sahip oldu. Evet her şey şu ana kadar iyi gitti diyebilirim. Peki ama javayı nerde hangi editorde yazabilirim. Araştırmam sonucunda her zaman olduğu gibi en güçlü en yetenekli en karizmatik editor ile bu işleri halledebileceğim sonucuna vardım. NOTEPAD. Tek yapmam gereken, uygulamayı notepad ile yazıp, java uzantısı ile kaydetmek. Daha sonra yazılan bu uygulamlayı bytecode�lara çevirmek için, javac (Java Compiler-Java Derleyicisi) programını kullanmak. Sonra ise, yazılımış olan uygulamayı başlatmak için java programı ile yazılan bytecode�ları çalıştırmak.

Aslında ilk uygulamamı yazmak için sabırsızlanıyorum. Kaynakların hemen hemen hepsi, yeni bir programlama dilini anlatırken ilk yazılan programı, Hello World yada Merhaba Dünya olarak adlandırırlar. Bende bu çizgide devam edeceğim. Çok sıradan olucak ama bir noktadan başlamam gerekiyor. Bu amaçla kendi bilgisayarımda java örneklerim için ayrı bir klasör açtım. Tüm örneklerimi burada yazıp geliştireceğim. Şimdi, aşağıdaki uygulama kodlarını notepad ile yazıp, JavaDunyam.java ismi ile kaydediyorum.

```
public class JavaDunyam
{
    public static void main(String[] args)
    {
        System.out.println("Java Dunayasına ilk adımımı attım
galiba...");
    }
}
```

Şimdi bu yazdığım kodu java derleyicisi ile derlemem gerekiyor. Hımmm. Peki ama benim java derleyicimin adresi nerede? Bir aramadan sonra kurmuş olduğum jsdk1.3.1_01 in klasörü içinde yer alan bin klasöründe, javac derleyici dosyasını buluyorum. Uygulamayı derlemek için önce bu klasöre komut satırından gittim. Sonuç tam anlamıyla hüsran hem bir hata mesajım var hemde bir programı derlemek için acaip bir yol kattetim.

```
Command Prompt
C:\jdk1.3.1_01\bin>javac c:\javasamples\javadunyam\javadunyam.java
c:\javasamples\javadunyam\javadunyam.java:1: class JavaDunyam is public, should
be declared in a file named JavaDunyam.java
public class JavaDunyam
       ^
1 error

C:\jdk1.3.1_01\bin>
```

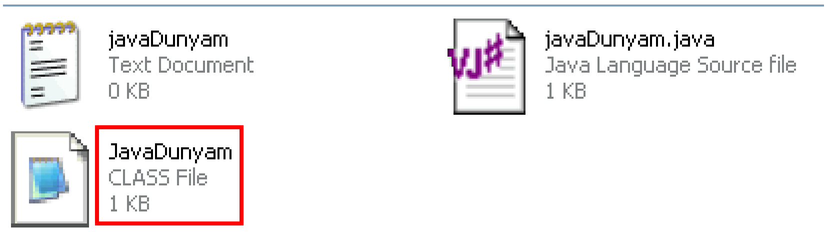
Öncelikle hatayı araştırmam gerekiyor. Şimdi gidip kendime bir kahve almanın tam sırası. Anlaşılan bu kahve molamız çok uzun sürecek. Araştırmam sonucu kitapların vardığı en önemli ortak nokta şu oldu. Java, C dili ve türevleri gibi büyük küçük harf duyarlıklı bir dil. Dolayısıyla buradaki hata mesajından çıkan sonuç sınıf adı ile java uzantılı dosya adının bire bir aynı olması. Dikkat ediyorumda da benim dosya adım javadunyam.java ama kullandığım sınıf adı JavaDunyam. Diğer taraftan bu kadar hassa bir noktayı iyice araştırmam gerektiğini düşünüyorum. Bu kez şöyle bir komut veriyorum.

```
Command Prompt
C:\jdk1.3.1_01\bin>javac c:\javasamples\javadunyam\JavaDunyam.java

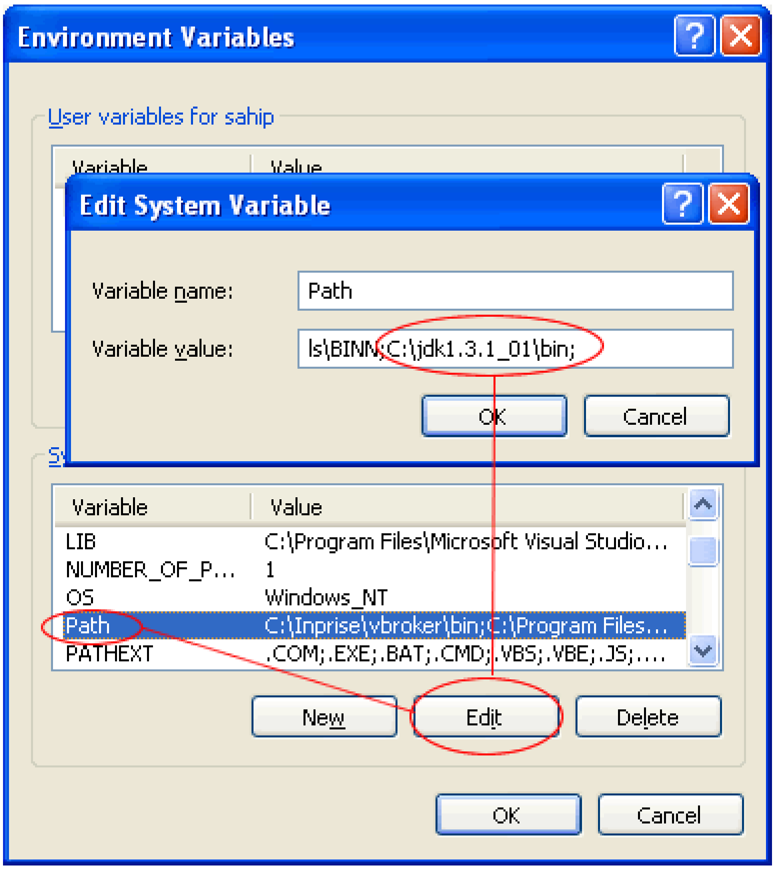
C:\jdk1.3.1_01\bin>_
```

Derleyiciye dosya adını sınıf adı ile aynı şekilde veriyorum Bu kez herhangibir hataya maruz kalmadan uygulamanın derlendiğini düşünüyorum. Kontrol etmenin tek yolu var o da klasörün içibe bakmak. Evet java uzantılı dosyam, java derleyicim sayesinde, class uzantılı bytecode dosyası haline

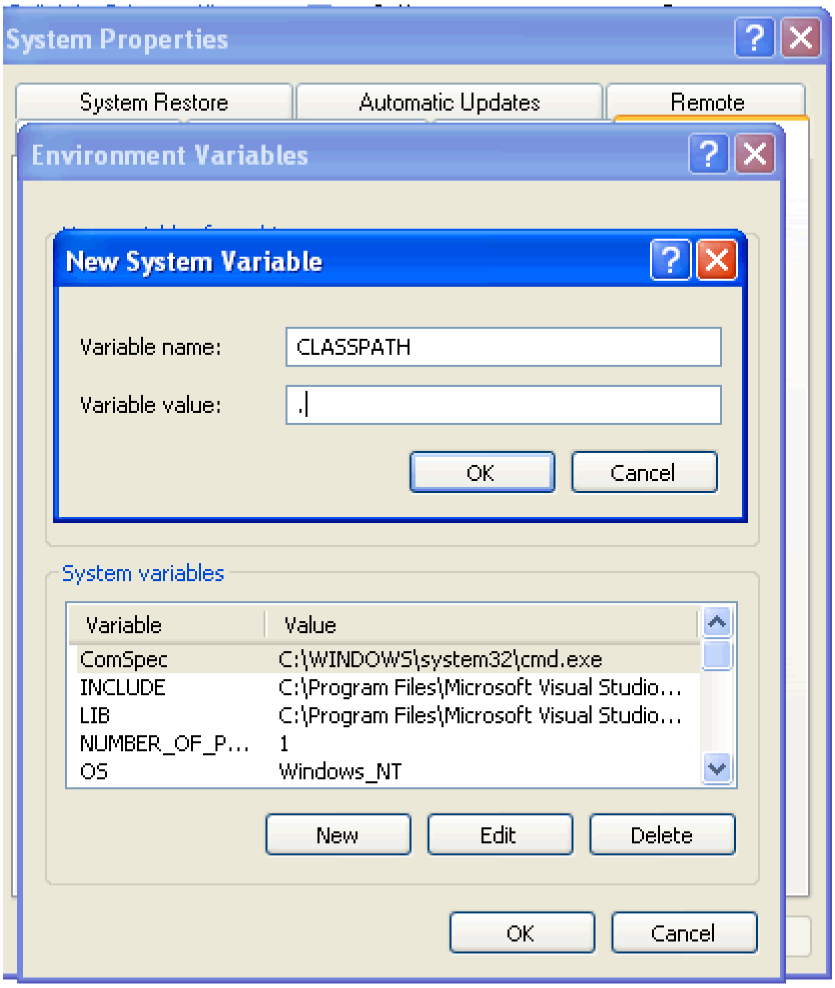
getirilmiş. Dikkat ettimde, dosya adı JavaDunyam. Yani sınıf adım ile birebire aynı. Burdan bir ders aldım, sınıf adım ile dosya adını birebir yazacağım bundan sonra.



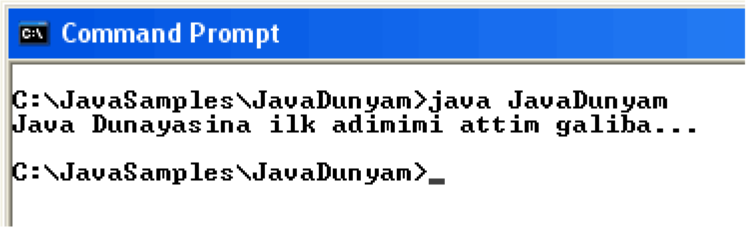
Ancak burada hoşuma gitmeyen bir nokta var. Ben her seferinde, bin klasörüne mi gideceğim? Bu çok saçma olur. Kaynak kitaplar, bu konu ile ilgili olarak, bin klasörününün systemdeki path tanımlamalarına eklenmesi gerektiğini söylüyorlar. Bunun nasıl yapıldığına baktığımda son derece kolay bir işlem olduğunu gördüm. Xp kullandığım için, windows�un bu versiyonu üzerinde, path tanımının nasıl girileceğini inceledim. Control Panel� den, System penceresine buradada Advanced kısmına buradada Environmet Variables kısmına girdim. Zaten sistemde yüklü olan programlar nedeni ile burada bir Path tanımlaması vardı. Bunu edit ederek sonuna, jsdk�in kurulduğu klasördeki bin dizinini girdim.



Artık aynı java uygulamamı bulunduğu klasör içinden rahatça derleyebiliyorum. Bununla birlikte kaynak kitaplar, birde CLASSPATH adı verilen bir değerin daha girilmesi gerektiğini söylüyorlar. Bu değeri, sisteme sonradan java uygulamalarını yüklediğimizde, gerekli sınıfların otomatik olarak aranıp bulunması için yapıyormuşuz. Doğruyu söylemek gerekirse ne demek istediklerini henüz tam olarak anlamış değilim ama sanıyorumki ileride ortaya çıkacaktır. İstedikleri gibi olsun diyerek, bu tanımlamayıda aynı şekilde gerçekleştirdim.



Artık gerekli ayarlamaları yaptığıma ve uygulamayı başarılı bir şekilde derlediğime göre nasıl çalıştığına bakmatada yarar var elbette. Çünkü eski deneyimlerim, program kodunun başarılı bir şekilde derlenmesi ile, doğru bir şekilde çalışmasının tamamen farklı kavramlar olduğun göstermiştir hep. Doğru şekilde derlenipte mantıksal programlama hataları nedeni ile pek çok istenmeyen sonuç doğduğunu çok iyi biliyorum. Diğer yandan yazdığım bu bir kaç satırlık kod derlendiğine göre mantıksal olarak bir hata olmayacağından neredeyse eminim. Java ile yazılmış uygulamalara ait bytecode dosyalarını çalıştırmak için, yine bin dizininde bulunan java isimli programı kullanıyoruz. Ama bu sefet bin klasörüne gitmemize gerek yok neyseki. Çünkü path tanımını bu iş için güncelledik. İşte sonuç. Mükemmmel, olağanüstü bir program demeyi çok isterdim. Ama sadece A harfinin tabanındayız desem yerinde olur herhalde.



Artık program kodlarını incelem zahmetine girebilirim. Kahvemim bitmesinede az kaldı. İlk satırdaki tanımlamamız aslında bana çok tanıdık geldi. Özellikle C# ile yazılan programlardan. Nitekim Java dilide nesne yönelimli bir dil ve sınıf kavramının önemi bu basit kodda bile görünüyor. Bu dildede her şey sınıflar üzerine kurulmuş durumda. Kaynaklarıma baktığımda, sınıflar ile ilgili detaylı anlatımların yer aldığını görüyorum. Sanıyorum ki, ilerliyen kahve molalarımda bu konulara değinebileceğim. Public tanımlaması anladığım kadarı ile bu sınıfın başka sınıflar içerisindende çağırılabileceği anlamına geliyor ki öyle. Daha tanıdık gelen başka bir satır ise main yordamının olduğu satır. Main yordamı java dilindede, programın başlangıç noktası olarak görülüyor. Nitekim kodumuzu buraya yazdık. Acaba kim kime benziyor bilemiyorum. Arada pek fark yokmuş gibi geliyor ama Microsoft her zamanki gibi benzerliği usta bir pazarlama stratejisi ile ortadan kaldırıyor. Lütfen M harfinin iki varyasyındada kullanımına dikkat edelim.

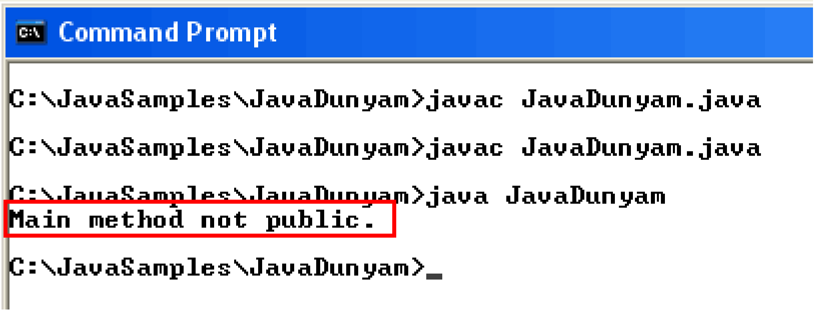
Java	**C#**
public static void main(String[] args)	static void Main(string[] args)

Diğer yandan System.out.println ifadesi ise " işaretleri arasında yazılmış olan metni, komut penceresine yazdırıyor. Bu ise C# dilindeki WriteLine ile aynı işleve sahip. Aslında buradaki kodlar biraz oynamak lazım. Eminimki yeni ufuklar açacaktır. Öncelikle main yordamındaki, public kelimesini kaldırmak istiyorum. Bu durumda kodum aşağıdaki gibi görünecek.

```
public class JavaDunyam
{
```

```
    static void main(String[] args)
    {
        System.out.println("Java Dunayasına ilk adımımı attım
galiba...");
    }
}
```

Yazdığım uygulamayı derlediğimde hatasız bir şekilde derlendiğini gördüm. Class uzantılı dosyam oluşturulmuştu. Öyleyse her şey yolunda görünüyordu. Ama programı çalıştırdığımda hiçte öyle olmadığını gördüm.



Main metodunun public olması gerektiği sonucuna hemen varabildim. Aslında işi biraz daha ileri götürmek lazım. Neden başka bir sınıf yazıp, JavaDunyam sınıfı içinden bu sınıfa ait bir nesne örneği yaratmıyor ve bu sınıf içindeki bir metodu çağırmıyorum. Ne kadar zor olabilirki. Sonuçta java ile program yazarken kendimi C benzeri bir dil ile yazıyormuşum hissine kapılıverdim nedense. İşte JavaDunyam.java isimli kaynak dosyamın yeni kodları.

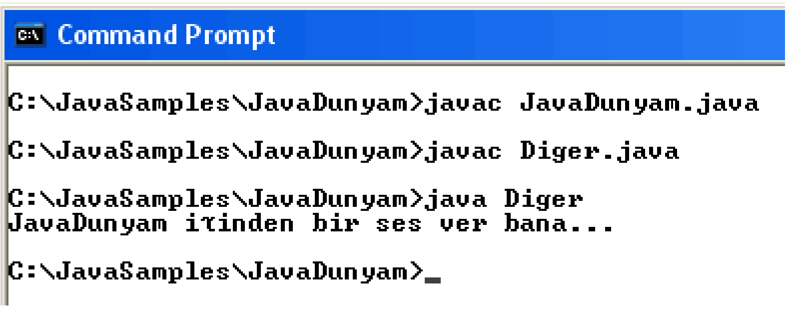
```
public class JavaDunyam
{
    public static void main(String[] args)
    {
        System.out.println("Java Dunayasina ilk adimimi attim
galiba...");
    }
    public void Deneme()
    {
        System.out.println("JavaDunyam içinden bir ses ver
```

```
bana...");
    }
}
```

Bu sınıf içinde Deneme isimli bir metod tanımladım. Amacım, bu metodu başka bir sınıf içinden çağırmak. Bu amaçla Diger isimli başka bir sınıfı Diger.java isimli dosya içinde aşağıdaki gibi tanımladım.

```
public class Diger
{
    public static void main(String[] args)
    {
        JavaDunyam jd=new JavaDunyam();
        jd.Deneme();
    }
}
```

Burada yaptğımı aslında C# ile yaptığım şeylerin aynısı. JavaDunyam sınıfı için bir nesne tanımladım. Sonrada nokta notasyonunu kullanarak, Deneme isimli metodu çağırdım. Şimdi bu kodları yeniden derleyip Diger isimli programımızı bir çalıştıralım. Bakalım herşey yolunda gidecek mi?



Daha güzel bir sonuç olamazdı. Sanıyorumki Java�yı öğrenmeye çalışırken çok zevk alacağım. Bakalım daha neler neler var. Kahvem bitmiş. Sanırım bir ara verme vakti. Bir sonraki kahve molasında bakalım neler olucak.

Bölüm 2:Alet Çantamızı Dolduralım

Geçtiğim hafta boyunca, Java ile ilgili kaynakları ve dökümanları incelemeye devam ettim. Her programlama dili için olmassa

olmaz gerekli olan bir takım temel bilgileri araştırıyordum. Bunlar bir programcı için, dilin temel kullanımında ihtiyacı olduğu materyallerdir. Genelde her programlama dili için bunlar gereklidir. Ancak elbette bunlar programlama dilleri arasında farklılık gösterebilir. Bahsettiğim konu, değişkenler, koşullu ifadeler ve döngüler. Bu kahve molasında bunları işlemeye çalışacağım.

Java programlama dilinde değişkenlerin neler olduğunu bir tablo halinde hazırladım. Genelde programlama dilllerinde bu tip değişken tipleri hep tablolar halinde sunulur. Değişkenleri sıkça kullandıkça, bunların alt sınır, üst sınır ve alan büyüklükleri gibi bilgileri zamanla unutabiliriz. Şahsen ben hep unuturum. Ancak programlarımızı hazırlarken nerede en uygun değişken kullanılır bunu da bilmek isteriz. Ben oldum olası bu tip tabloları ezberleyemem. Zaten ezberleme taraftarı değilim. O nedenle bir sürü not defterim vardır ve taşıdığım çanta genelde ağır olur. Java dilinde kullanılan değişkenler içinde aynı şeyleri hissediyorum. Sanıyorum ki bir tablo hazırlayacağım ve bunun güzel bir karton baskısını yanımda taşıyacağım.

Esasen Java dilinde, değişkenler, temel veri tipleri olarak anılırlar. Bu anlamda Java'da iki veri tipi olduğunu söyleyebiliriz. Değişkenlerin tanımanması için kullanılan Temel Veri Tipleri(Primitive) ve nesnelerin tanımlanması için kullanılan Referans Tipleri. C# dilinde'de bu böyledir zaten. Temel veri tipinden elde edilen değişkenler, bellekte yığın adı verilen bir bölgede tutulurlar. Oysa referans tiplerin tutuluş şekli daha farklı. Referans tipinden bir nesne, sahip olduğu üyülerin tutulduğu bellek bölgesindeki alanların başlangıç adresine işaret ederki bu tip nesneler yığında tutulurken, sahip oldukları içerik öbekte tululur. Java programlama dilinde Temel Veri Türleri aşağıdaki tabloda olduğu gibidir.

Veri Tipi	**Alan Büyüklüğü**	**Kategori**
byte	8 bit	Tamsayı Tipleri
short	16 bit	
int	32 bit	

long	64 bit	
float	32 bit	Kesirli Sayı Tipleri
double	64 bit	
char	16 bit	Karakter Tipi
boolean	-	Mantıksal Tip (ture/false)

Görüldüğü gibi temel veri tipleri bunlar. Aslında bu tabloda birde alt aralık ve üst aralık bilgilerinin olması gerekiyordu. Ancak bunlar genelde kaynaklarda üstsel bilgi olarak gösterilmiş. Gerçekte, bu veri tiplerinin en üst ve en alt sınır bilgilerini küçük bir program kodu yazarakta öğrenebilirim. İşte küçük programım.

```
public class Sinirlar
{
    public static void main(String[] args)
    {
        System.out.println("Integer veri tipi");
        System.out.println("Integer alt sınır
:"+Integer.MAX_VALUE);
        System.out.println("Integer ust sınır
:"+Integer.MIN_VALUE);
        System.out.println("---");

        System.out.println("Double veri tipi");
        System.out.println("Double alt sınır
:"+Double.MAX_VALUE);
        System.out.println("Double ust sınır
:"+Double.MIN_VALUE);
        System.out.println("---");

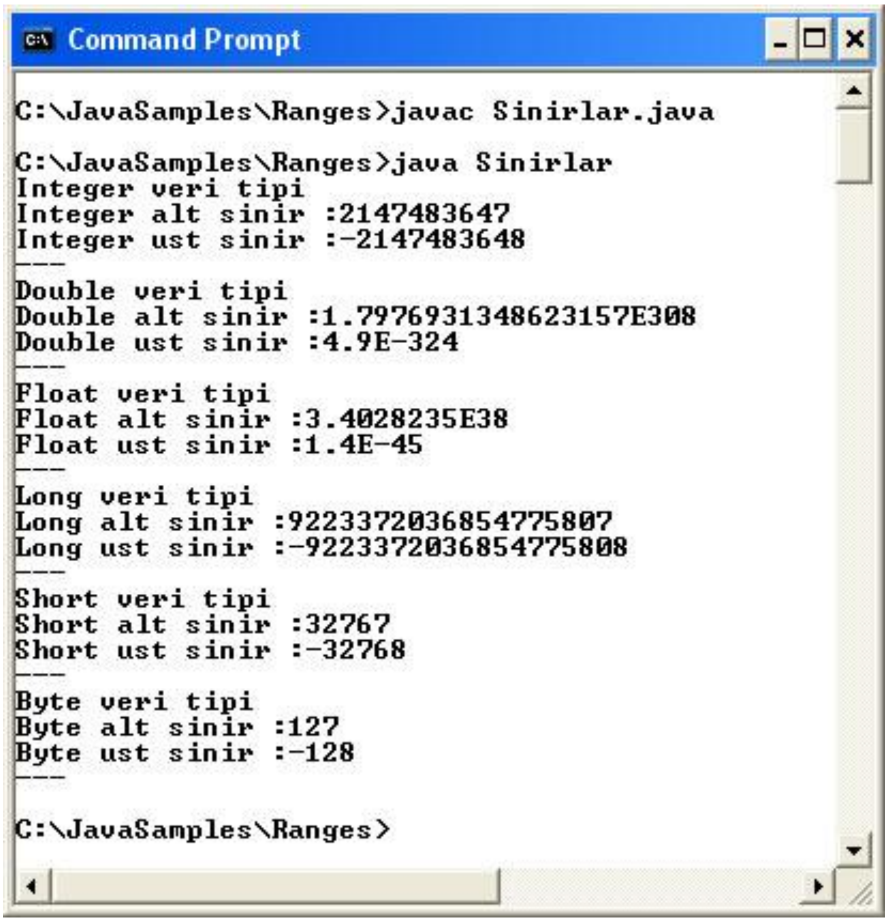
        System.out.println("Float veri tipi");
        System.out.println("Float alt sınır :"+Float.MAX_VALUE);
        System.out.println("Float ust sınır :"+Float.MIN_VALUE);
        System.out.println("---");
```

```
        System.out.println("Long veri tipi");
        System.out.println("Long alt sınır :"+Long.MAX_VALUE);
        System.out.println("Long ust sınır :"+Long.MIN_VALUE);
        System.out.println("---");

        System.out.println("Short veri tipi");
        System.out.println("Short alt sınır :"+Short.MAX_VALUE);
        System.out.println("Short ust sınır :"+Short.MIN_VALUE);
        System.out.println("---");

        System.out.println("Byte veri tipi");
        System.out.println("Byte alt sınır :"+Byte.MAX_VALUE);
        System.out.println("Byte ust sınır :"+Byte.MIN_VALUE);
        System.out.println("---");
    }
}
```

Şimdi yazdığım uygulamayı Java Derleyicisi ile derliyorum. Hayret, her hangibir hata vermeden derledi. Daha sonrada programı çalıştırıyorum. Sonuç gayet güzel oldu. Tam istediğim gibi, temel veri türlerine ait alt ve üst sınır değerlerini elde etmeyi başardım. Bunu yaparken, bu veri tipleri için Java içinde geliştirilmiş hazır sınıfları kullandım. Örneğin int veri tipi için Integer sınıfını kullandım. Aslında tüm temel veri tipleri için sınıflar mevcut. Tek yaptığım bu sınıfların MAX_VALUE ve MIN_VALUE özelliklerinin değerlerini ekrana yazdırmak oldu. Bu kadar basit.

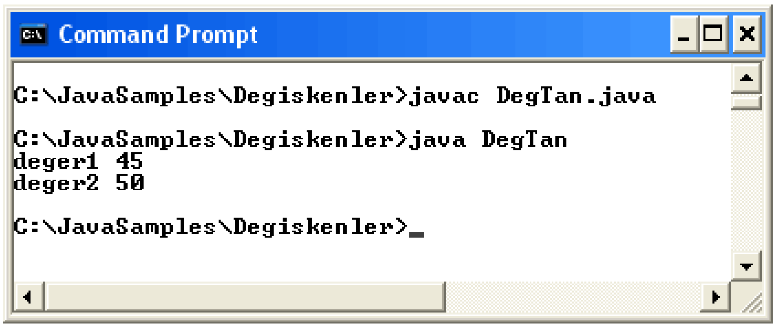


Burada aslında önemli bir noktada her veri tipi için bir sınıfın var olması. Dolayısıyla bir değişken tanımlamasını iki şekilde yapma imkanına sahibim. Örneğin aşağıdaki kod parçasında, int (integer-tamsayı) veri tipinden iki değişkenin farklı şekillerde tanımlandığını görüyoruz.

```
public class DegTan
{
   public static void main(String[] args)
   {
      int deger1=45;
      Integer deger2=new Integer(50);

      System.out.println("deger1 "+deger1);
      System.out.println("deger2 "+deger2);
   }
}
```

Bu uygulamayı çalıştırdığımda aşağıdaki sonucu elde ettim.



Sonuçta iki tane integer tipinde değişken tanımlamıştım, ancak bu tanımlamalar arasında büyük farklar olduğuna inanıyorum. Şimdi bunu araştırmam gerektiğini düşünüyorum. İlk başta gözüme çarpan, Integer sınıfını kullanarak, integer tipte veriyi barındıran Deger2 isimli bir nesne tanımlamamız. Bu durumda deger2 değerinin bellekte tutuluş şekli, deger1'den farklı olmaktadır. Diğer yandan tüm temel veri tiplerinin birer sınıfı mevcuttur ve bu sınıflar java.lang adı verilen bir pakette bulunmaktadırlar. Kaynaklardan edindiğim bilgiye göre, burada adı geçen paket kavramının, C#'taki namespace (ad alanı) kavramı ile aynı olduğu sonucuna vardım. Bu bilgiye ulaşmak benim için internette biraz zaman harcamak ile gerçekleşti. Java için herzaman elimizin altında bulunması gereken yardım dokumantasyonu olan Java 2 Platform Std.Ed. Documentation v1.3.1' i

http://java.sun.com/j2se/1.3/docs.html

adresinden indirdim. Bu döküman yaklaşık olarak 22 megabyte'lık bir zip dosyası. Açıldığında, html dokumatasyonuna ulaşabiliyorsunuz. Burada aradığım hemen her konuya ait bilgi mevcut. Ancak bazı konuların yanında web pages yazılı. Bunlar internetten online olarak bakılabilecek yada indirilebilecek adreslere işaret ediyor. İşte, java.lang paketinin içeriğinede bu dokmantasyondan ulaştım. Temel veri tiplerine ait sınıfları kullanarak pek çok fonksiyonu kullanma şansına sahip olduğumu gördüm. Örneğin, ilk örneğimizde kullandığımız MAX_VALUE ve MIN_VALUE özellikleri gibi. Yada integer değeri String'e dönüştürmek için kullanılan toString metodu ve daha pek çok sayısız metod yer alıyor.

Bir diğer konuda, Java'daki temel veri türünden değişkenler ile, referans türünden nesneler arasındaki temel farklılıklar. Burada

önemli olan konu, bu iki veri türününde bellekte farklı şekillerde tutuluyor olması. Temel veri türünden olan değişkenler bellekte kendi isimleri ile stack(yığın) adı verilen bölgede tutuluyorlar. Referans tipinden olan nesneler ise, belleğin heap(öbek) adı verilen bölgesinde tutuluyor. Tanımlanan referans tipindeki nesne, öbekte yer alan verilerine işaret eden ve yığında tutulan bir değişken adına sahip oluyor. Bu kavramlar aslında karmaşık gibi görünsede, nesneler arasındaki atamalarda önemli sonuçlar doğuruyor. Bunu açıklamak için örnekler geliştirmek sanıyorumki en doğrusu olucaktır. Bu örnekte yapmak istediğim, bir sınıf hazırlamak ve bu sınıf içinde değişkenler tanımlamak. Sonra bu sınıfa ait bir nesne örneği yaratacak ve onu aynı sınıftan türetilen başka bir sınıfa atayacağım. Bu işlemlerin gerçekleşmesi halinde bellekteki oluşucak hareketleri ise şekille göstermeye çalışacağım. Öncelikle örneğimi geliştireyim. İlk önce nesnelerim için kullanacağım sınıfımı oluşturuyorum.

```
public class Sinifim
{
   public int Deger1;
   public int Deger2;

   public Sinifim()
   {
      Deger1=0;
      Deger2=0;
   }

   public Sinifim(int d1,int d2)
   {
      Deger1=d1;
      Deger2=d2;
   }

   public void Yaz()
   {
      System.out.println("Deger1 :"+Deger1);
      System.out.println("Deger2 :"+Deger2);
   }
```

```
}
```

Doğruyu söylemek gerekirse bu kodlar bize çok şey söylüyor. Buradaki kodları tamamen C#'taki bilgimi kullanarak yazdım. Örneğin sınıfı tanımlaması içinde, iki adet yapıcı (constructor) metod var. İlki Deger1 ve Deger2 isimli değişkenlerimize 0 değerlerini atıyan parametre almayan ve sınıflar için varsayılan olarak kabul edilen yapıcı metodumuz. Diğeri yapıcı metodumuz ise, bu değişkenlere aldığı parametre değerlerini atıyor. Birde Yaz isimli bir metodumuz var. Bu metod ise, sınıfımızın Deger1 ve Deger2 isimli integer değişkenlerini komut satırına yazdırıyor. Yapıcı metodların kullanımı, sınıflar içindeki yeri gibi kavramlara şimdilik göz ucu ile bakmış oldum. Bu kavramları ve daha fazlasını diğer kahve molalarımda incelemek istiyorum. Hazır kahve demişken, kahvemi tazelesem iyi olucak sanırım. Gelelim diğer sınıfımıza. Bu sınıfımız ise Sinifim sınıfı türünden nesneler üzerinde işlem yapmak için kullanılıyor.

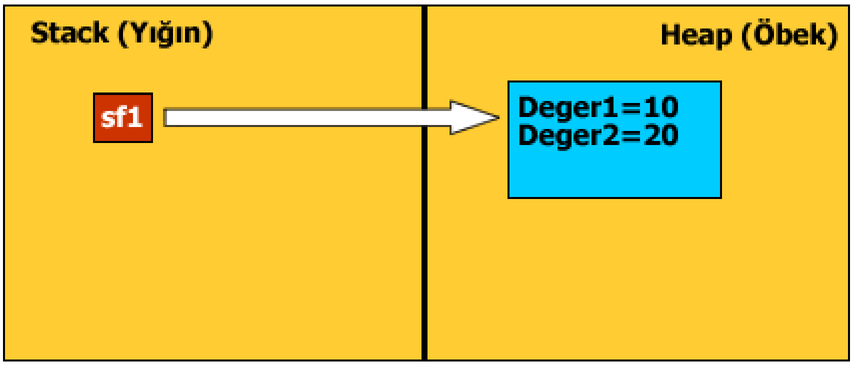
```
public class Program
{
    public static void main(String args[])
    {
        Sinifim sf1=new Sinifim(10,20);
        System.out.println("Sf1 nesnesi için");
        sf1.Yaz();

        Sinifim sf2=new Sinifim();
        System.out.println("Sf2 nesnesi için");
        sf2.Yaz();

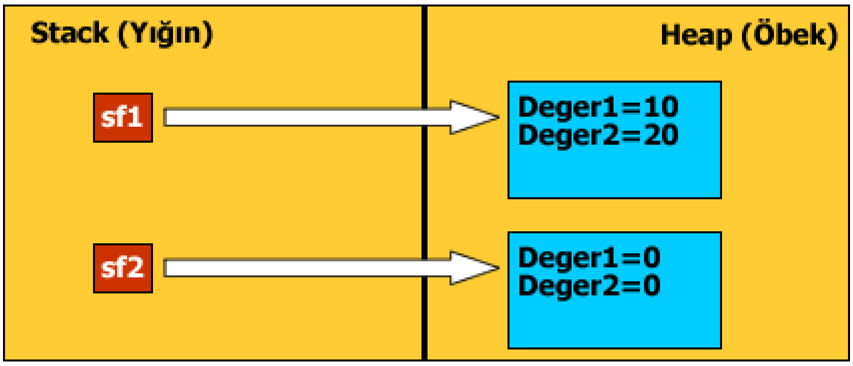
        System.out.println("sf1 nesnesi, Sf2 nesnesine
aktarılıyor");
        sf2=sf1;
        sf2.Yaz();

    }
}
```

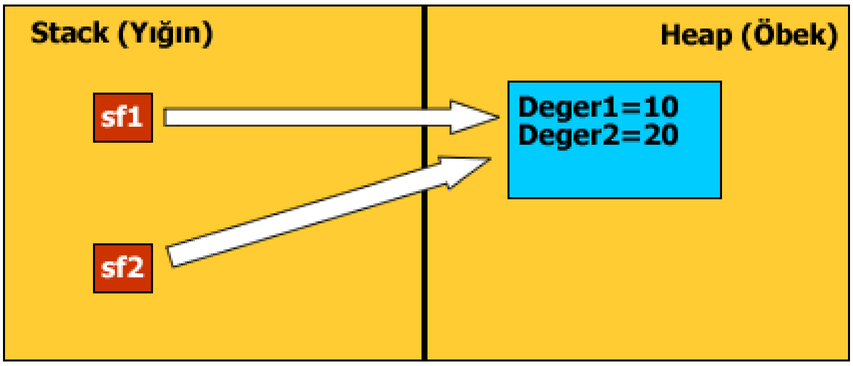
Bu kodlarla göstermek istediğim, referans tipleri arasındaki atamalar sonucu oluşan ilişkidir. Program içinde, önce sf1 isimli bir nesne örneği oluşturuluyor ve bu nesne içindeki Deger1 ve Deger2 integer değişkenlerine 10 ile 20 değerleri atanıyor. Bu durumda belleğin durumunun tasviri aşağıdakine benzer olucaktır.



Daha sonraki adımda ise, sf2 isimli başka bir Sinifim nesne örneğini oluşturuyoruz. Bu kez nesneyi, Sinifim sınıfının varsayılan yapıcı metodu ile oluşturuyor ve değişkenlerimize 0 değerini atıyoruz. Bu halde, belleğin durumu şu şekilde olucaktır. Bellekte Sinifim, sınıfı türünden iki adet nesne örneği mevcuttur.



Buraya kadar herşey normal. Ancak son adımda, sf1 , nesnesini sf2 nesnesine atıyoruz. İşte bu durumda olay biraz daha değişik bir hal oluyor. Nitekim, sf1 nesnemiz artık sf2 nesnemize işaret ediyor.



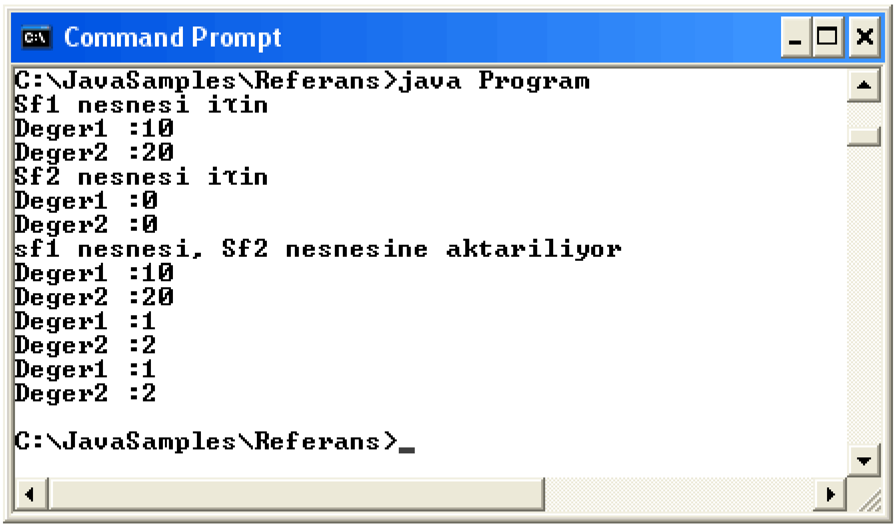
Şimdi örneğimizi biraz değiştirelim ve Sinifim sınıfına aşağıdaki DegerAta isimli metodu ve kod satırlarını ekleyelim.

```
public void DegerAta(int d1,int d2)
{
   Deger1=d1;
   Deger2=d2;
}
```

Şimdide Program sınıfına aşağıdaki kod satırlarını ekleyelim.

```
sf1.DegerAta(1,2);
sf1.Yaz();
sf2.Yaz();
```

Bu haldeyken, Program dosyasını çalıştırdığımda, her iki nesne örneğininde aynı değerleri yazdırdığını görürüz. Bu şu anlama geliyor. Atama işleminden sonra, öbekteki aynı bölgeyi işaret eden bu iki nesneden birisinin içindeki değerlerin değiştirilmesi, diğer nesneninde aynı değişiklikleri işaret etmesi anlamına gelmektedir. Sonuç aşağıdaki gibi olucaktır.



Değişkenler arası atamalara gelince. Burada durum referans tiplere göre daha farklı. Çünkü değişkenler oluşturuldukları isim ile bellekte tutulur. Bu konuyuda bir örnek üzerinde incelemek taraftarıyım. Kısa ve basit bir örnek bize yeterli olucaktır sanırım.

```
public class Program2
{
    public static void main(String args[])
    {
        int Deger1=50;
        int Deger2;

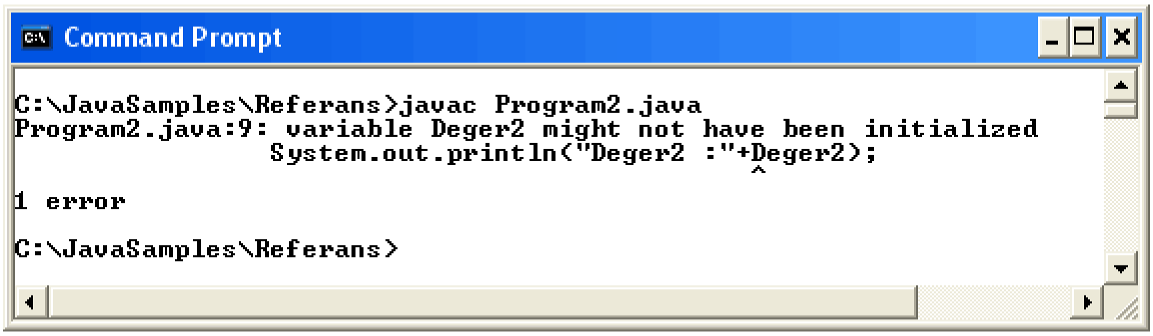
        System.out.println("Deger1 :"+Deger1);
        System.out.println("Deger2 :"+Deger2);

        Deger2=Deger1;
        System.out.println("Deger1 :"+Deger1);
        System.out.println("Deger2 :"+Deger2);

        Deger1=48;
        System.out.println("Deger1 :"+Deger1);
        System.out.println("Deger2 :"+Deger2);

    }
}
```

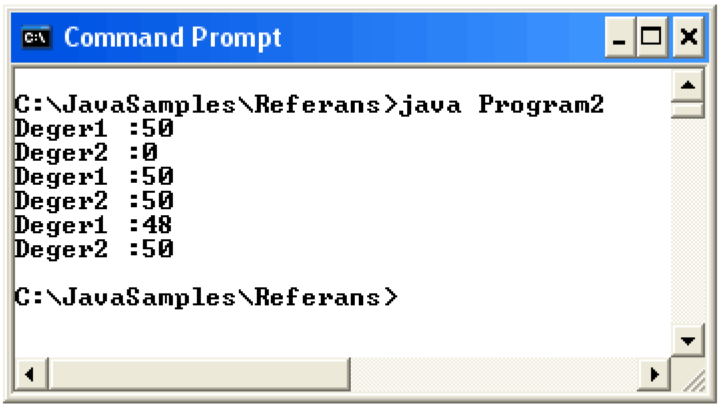
Kodu derlediğimde hiçte beklemediğim bir hata ile karşılaştım.



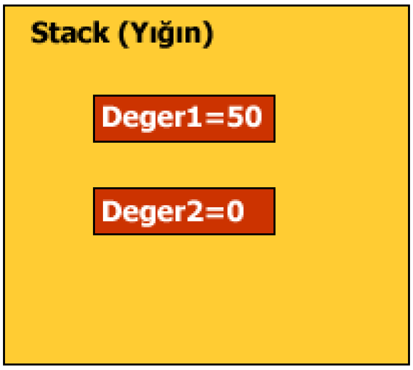
Sanıyorumki hatanın sebebi Deger2 isimli integer veri tipindeki değişkene ilk değer atamasını yapmamış olmamdı. Bu durumda kodun bu satırını aşağıdaki gibi değiştirdim ve programın başarılı bir şekilde derlendiğini gördüm.

int Deger2=0;

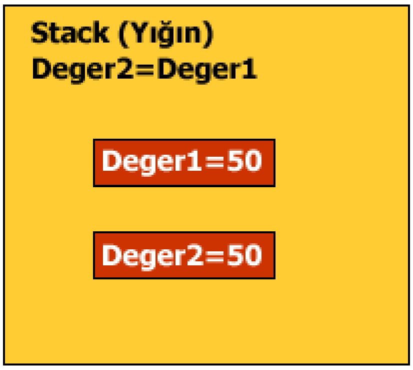
Şimdi uygulamamı çalıştırdığımda aşağıdaki sonuçlar ile karşılaştım.



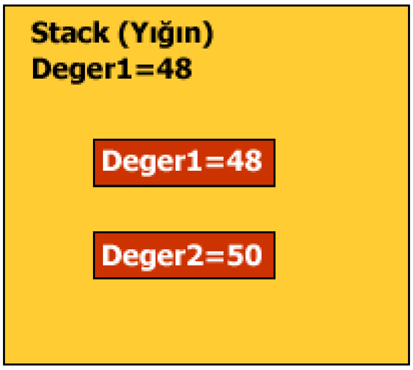
Şimdi kodu güzelce bir incelemem gerektiğini düşünüyorum. Kahvemden bir yudum aldım ve Javaca düşünmeye başladım. İlk olarak Deger1 ve Deger2 değişkenlerimizi tanımlıyor ve bunlara ilk değer olarak sırasıyla 50 ve 0 değerlerini atıyoruz. Bu noktada belleğin durumu aşağıdaki tasvirde olduğu gibi olucaktır. İki ayrı integer tipte değişken belleğin yığın bölgesinde yerlerini almıştır.



Daha sonraki adımda ise, Deger1 değişkeninin değeri Deger2'ye atanıyor. Bu durumda Deger1'in sahip olduğu 50 değeri, Deger2 değişkenine atanmış ve belleğin görünümü aşağıdaki şekilde olmuş oluyor.



Son adımda ise, Deger1' değişkenine 48 değerini atıyoruz ve bellekteki Deger1 değişkeninin değerinin değişmesine neden oluyoruz. Referans tiplerinde olduğu gibi, değişken tanımlamalarında, atamadan sonraki değişiklikler, değişkenlerin birbirlerini etkilemesine neden olmamaktadır.



Sanıyorumki değişken tipleri ve referans tipleri arasındaki farkları daha iyi anladık. Evet alet çantamızı doldurmaya devam edelim. Bizim için gerekli olan diğer önemli ve hatta çok önemli unsurlar koşullu ifadeler ve döngülerdir. Bu her programlama

dili için böyledir. Öncelikle karşılaştırma ifadelerine bir göz atmak istiyorum. Karşılaştırma ifadelerimiz bana nedense çok tanıdık geliyor. If ve Switch. Bu ifadelerin kullanımını örnekler ile incelemek taraftarıyım. Öncelikle if koşul ifadesini incelemek istiyorum.

```
public class Kosullar
{
    public static void main(String args[])
    {
        int Not;
        Not=49;
        if(Not>50)
        {
            System.out.println("Sınıfı geçtin...");
        }
        else if((Not>45)&&(Not<50))
        {
            System.out.println("Kanaat kullanmalımıyım
bakayım?");
        }
        else
        {
            System.out.println("Sınıfta KALDIN?");
        }
    }
}
```

Aslında her şey çok açık ve ortada. If koşulları parantezler içinde yazılır ve karşılaştırma sonucu true ise hemen izleyen { } kod bloğu içindeki kodlar çalıştırılır. Eğer bu şart false ile sonuçlanır yani gerekli koşul sağlanmaz ise, varsa bir sonraki else koşuluna geçilir. Burada kullanmış olduğumuz bir takım karşılaştırma operatörleride mevcut. Java'da kullanılan karşılaştırma operatörlerü aşağıdaki tabloda yer almaktadır.

Karşılaştırma Operatörü	**Örnek**	**Açıklama**

==	(Deger1==Deger2)	Deger1, Deger2'ye eşit ise true, değilse false döndürür.
!=	(Deger1!=15)	Deger1 15'ten farklı ise true, değilse yani 15'e eşit ise false döndürür.
<	(Deger1<50)	Deger1 50'den küçükse true, büyükse false döndürür.
<=	(Deger1<=50)	Deger1 50'ye eşit veya küçük ise true, 50'den büyük ise false döndürür.
>	(Deger1>50)	Deger1 50'den büyükse true, küçükse false döndürür.
>=	(Deger1>=50)	Deger1 50'ye eşit veya büyükse true, 50'den küçükse false döndürür.

Diğer yandan ifadelerimizin birisinde && kullandık. Bu mantıksal VE anlamına gelen bir operatördür. Bu tip mantıksal operatörleri, iki veya daha fazla koşulun sonuçlarını bir arada

değerlendirmek istediğimizde kullanırız. Bu örneğimizde mantıksal operatörümüz, belirtilen iki şartında doğru olması şartıyla izleyen kod bloğundaki kodları çalıştırmaktadır. Java'da kullanılan mantıksal operatörler aşağıdaki gibidir.

Birinci Değer	**İkinci Değer**	**&& (Ve)**	**|| (Veya)**	**^ (Yada)**	**! (Değili) (Birinci Değere Göre)**
true	false	false	true	true	false
false	true	false	true	true	true
false	false	true	false	false	
true	true	true	true	false	

Kaynaklardan mantıksal operatörlerden && ve || için farklı bir kullanım tarzı daha olduğunu öğrendim. Buda & ve | operatörleri. Bu iki kullanım tarzı arasındaki farkı incelediğimde oldukça işe yarar bir sonuçla karşılaştım. && ve || operatörleri, her iki koşulunda işleme sokulmasını sağlar. Ancak bazen ilk koşulun true olması yada false olması sonucu belirlemek için yeterlidir. Böyle bir durumda her iki koşulu da karşılaştırmaktansa, yani koşul ifadesindeki tüm kodu çalıştırmaktansa, & ve | operatörlerini kullanarak, sadece soldaki koşula bakılmasını ve buna göre karar verilmesini sağlayabiliriz. Gelelim switch koşullu ifadesinin kullanımına. Bunuda yine bir örnekle incelemek en güzeli.

```
public class Kosullar
{
    public static void main(String args[])
    {
        int Deger1=3;

        switch(Deger1)
        {
            case 1:
            {
                System.out.println("Birinci sınıf nesnesi oluştur.");
                break;
```

```
            }
            case 2:
            {
               System.out.println("Ikinci sınıf nesnesi oluştur.");
               break;
            }
            case 3:
            {
               System.out.println("Ucuncu sınıf nesnesi oluştur.");
               break;
            }
            default:
            {
               System.out.println("Ana menuye don.");
               break;
            }
         }
      }
   }
```

Aslında buraya kadar herşey çok hızlı gelişti. Nitekim kullanılan materyaller C dilinden gelmekte ve C# dili içindede aynen kullanılmakta. Bu nedenle bir C# programcısı için Java'yı öğrenmek veya bir Java programcısı için, C# dilini öğrenmek hızlı oluyor diyebilirim. Neyse kahvemizi yudumlamaya devam edelim.

Sırada döngüsel ifadeler var. Bazen bir işlemi birden fazla sayıda uygulamak isteyeceğimiz durumlar olabilir. Gauss eğer bir bilgisayara sahip olsaydı inanıyorum ki 1'den 100'e kadar olan sayıların değil 1'den 1000'e kadar olan sayıların 4ncü dereceden kuvvetlerinin toplamını bulan bir uygulama geliştirir ve bizi Gauss formüllerini ezberlemekten kurtarırdı. Sanıyorum programlama dillerini geliştirenler Gauss'un çektiklerinden çok, okul yıllarında matematik sınavlarındaki Gauss formüllerini hatırlayamamaktan çekmişler. Burada, for döngüsü kullanarak bu işlemin nasıl gerçekleştiğini incelemeye çalışacağım. Burada üs almak için, Java içinde yer alan Math sınıfına ait pow isimli metodu kullandım.

```
public class Dongu
{
    public static void main(String args[])
    {

        double Toplam=0;

        for(int i=1;i<=1000;i++)
        {
            Toplam=Toplam+(Math.pow(i,4));
        }

        System.out.println(Toplam);
    }
}
```

For döngüsü dışında iki döngü çeşidi daha vardır. Bunlar while döngüleridir. İki çeşit while döngüsü vardır. Bunlardan birisi, ilk başta koşulu kontrol eder ve koşul sağlanıyorsa döngü içindeki kodları çalıştırır. Diğer çeşidinde ise, döngü içindeki kodlar en az bir kere çalışır ve koşul sonradan kontrol edilir. Her iki döngüde koşullar sağlandığı sürece devam eder. Örneğin yukarıdaki uygulamamızı while döngüsü kullanarak gerçekleştirelim.

```
Toplam=0;
int i=1;
while(i<=1000)
    {
        Toplam=Toplam+(Math.pow(i,4));
        i++;
}
System.out.println(Toplam);
```

While döngümüzün diğer şekli ilede bu döngüyü yazabiliriz.

```
Toplam=0;
int j=1;
do
```

```
{
   Toplam=Toplam+(Math.pow(j,4));
   j++;
}
while(j<=1000);
System.out.println(Toplam);
```

Artık hakiki bir mola vermenin zamanı geldi sanırım. Bu kahve molasında, alet çantamızı iyice doldurduk. Ancak bu çantaya doldurmamız gereken daha çok parça olduğu kesin. İlerleyen kahve molalarında Java'yı incelemeye devam edeceğiz.

Bölüm 3: Diziler

Şuanda sabahın saat 3 buçuğu ve yeni hazırladığım sıcak kahvemi yudumlarken, Java ile ilgilenmekten saatın ne kadar geç olduğunun farkına bile varamadım. Geçen hafta boyunca, Java programlama dilinin temellerini incelemeye devam ettim. Doğruyu söylemek gerekirse, sıkıcı olan bir kaç konu inceledim. Bunlardan birisi dizilerdi.

Hemen her programlama dilinde olduğu gibi dizilerinde java�da sıkıcı bir yeri var. Sıkıcı olmasını nedeni belkide, veri saklama ile ilgili olarak günümüz teknolojilerinin getirdiği veritabanı, xml, gelişmiş koleksiyonlar gibi kavramların çok daha fazla tercih ediliyor olması. Ancak ne olursa olsun, hangi programlama dili olursa olsun ve kavramları ne kadar sıkıcı olursa olsun, bunları bilmekte, incelemekte fayda var.Bu sabah saatlerindeki kahve molamda, dizilerin Java programlama dili içindeki yerini incelemeye çalışacağım. Bir diziyi belkide en güzel şu şekilde tanımlayabiliriz.

Dizi, belli bir sayıda ve aynı veri türünden değişkenlere, aynı isim altında erişilmesini sağlayan bir referanstır.

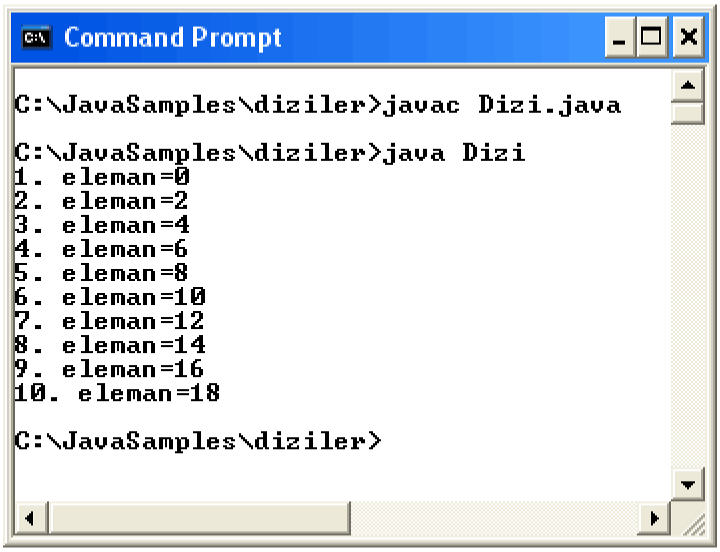
Aslında bu tanım içerisinde çok geniş kavramlar var ve açıklanması gerekli. Herşeyden önce, dizilerin referans olduklarını söylüyor. Java dilindede, C#�ta olduğu gibi diziler referans tipli verilerdir. Dolayısıyla bir dizi, belleğin yığın (stack) kısmında tutulurken, dizi elemanlarının tamamı, öbek

(heap)�te yer alır. Yani dizinin adı, öbekteki elemanların başlangıç adresine işaret eder. Bu dizilerin en önemli unsurlarından birisidir. Diğer yandan, diziler aynı veri türünden elemanlara sahip olmalıdır. Bununla birlikte dizilerin eleman sayısıda oluşturulurken belirlenir. Dolayısıyla, bir diziyi en baştan boyutlandırdığımızda, beliltilen sayıda elemanı aktarabiliriz ve buda dizinin boyutu ile dinamik olarak oynamamızı engeller. Doğruyu söylemek gerekirse bu kısıtlamalar belkide C# dilinde koleksiyonları geliştirilmesine ve daha çok önem kazanmasına neden olmuştur. Kimbili belki Java dilinde�de koleksiyonlar var. Şimdilik karşıma çıkmadılar.

O halde işe referans veri tipi olan dizilerin nasıl tanımlanacağından başlamak gerekiyor. Referans tipli olması, dizilerin oluşturulurken new yapılandırıcı kullanılarak oluşturulması gerektiği sonucunu ortaya çıkartıyor. Nitekim, diziler referans tipli nesnelerdir. Dolayısıyla bir sınıf örneği olmak zorundalar. Ancak diğer yandan, bir diziyi oluştururken new yapılandırıcısını kullanmak zorunda değiliz. Dizi elemanlarına, 0 tabanalı bir indis sistemi ile erişebilmekteyiz. Bu anlamda dizilerin, döngüler ile kardeş olmalarını gayet iyi anlayabiliyorum. Şimdi dizi kavramını incelemek için basit ama hızlı bir örnek geliştirmek düşüncesindeyim. Kahvemden bir yudum alıyorum ve aşağıdaki örneği yazıyorum.

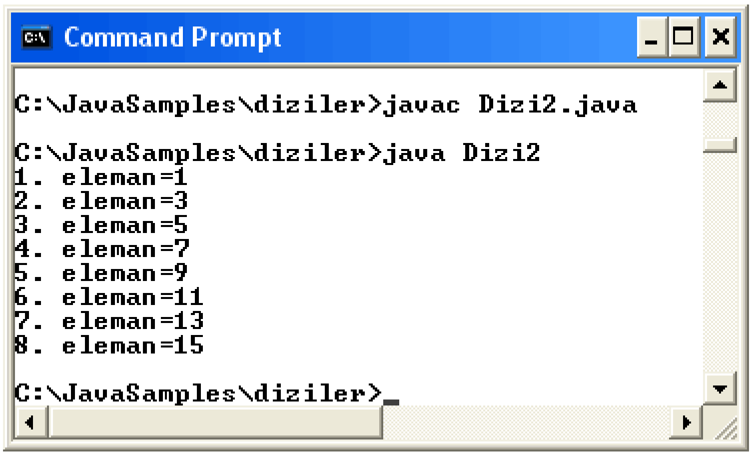
```
public class Dizi
{
    public static void main(String[] args)
    {
        int dizi1[]=new int[10];
        for(int i=0;i<dizi1.length;++i)
        {
            dizi1[i]=i*2;
           System.out.println((i+1)+". eleman="+dizi1[i]);
        }
    }
}
```

Uygulamayı başarı ile derledikten sonra çalıştırdığımda aşağıdaki sonucu aldım.

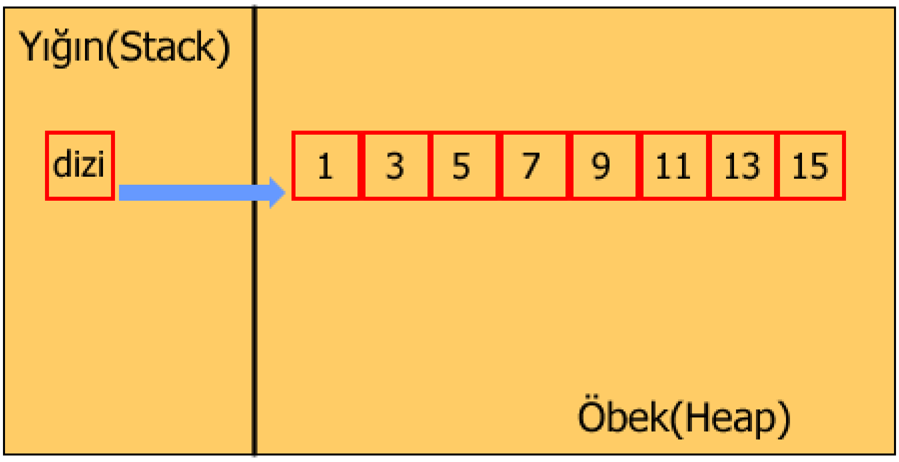


Kodların çalışma sistematiği oldukça basit. Diziyi tanımlaka için new yapılandırıcısını kullandık ve dizinin integer veri tipinden 10 elemanlı olmasını sağladık. Bu noktadan sonra dizimize elemanlar atamak için bir döngü kullandık. Bu döngü dizimizin boyutuna kadar hareket eden ve her bir elemana indis değerinin iki katını aktaran bir yapıda. Kullandığımız length özelliği, dizilerin boyutunu elde etmekte kullanılmakta. Buradanda zaten, dizilerin bir Java sınıfına ait olduğu sonucuna varabiliriz. Kodumuzda, dizi elemanlarını okumak içinde aynı döngü içindeki dizi1[i] şeklindeki yazımı kullandık. Dizileri yukarıdaki örnekte olduğu gibi new yapılandırıcısı ile tanımlayabileceğimiz gibi, başka alternatif bir yoluda kullanabiliriz. Benzer bir örnek uygulam aşağıda yer alıyor.

```
public class Dizi2
{
    public static void main(String[] args)
    {
        int[] dizi={1,3,5,7,9,11,13,15};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```



Burada diziyi tanımlarken elemanlarınıda baştan atamış olduk. Bu aynı zamanda verdiğimiz eleman sayısına göre dizinin otomatik olarak boyutlandırıldığınıda göstermektedir. Son örneğimizin bellekteki yerleşim planına göz atarsak aşadıdaki tasvire benzer bir yapının oluştuğunu görürüz.

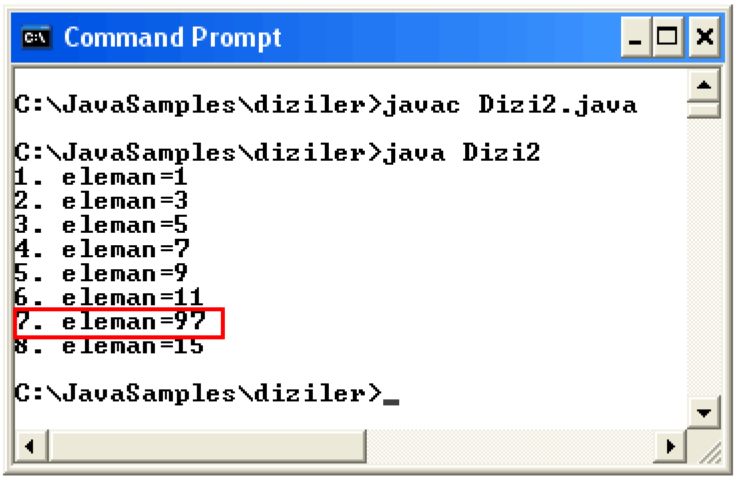


Peki ya bir diziye tanımlandığı veri tipinden başka bir tipte eleman eklemeye kalkarsak ne gibi sonuçlarla karşılaşabiliriz. Doğrusu bunu merak ediyorum. Bu nedenle yukarıdaki örnekte, integer tipteki dizimize, başka bir veri tipinden eleman eklemeye karar verdim. Bu amaçla, �a� karakterini sincizce dizi elemanları arasına yerleştirdim.

```
public class Dizi2
{
    public static void main(String[] args)
    {
        int[] dizi={1,3,5,7,9,11,�a�,15};
```

```
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```

Uygulamayı derlediğimde her hangibi hata mesajı ile karşılaşmadım. Diğer yandan uygulamayı çalıştırdığımda aşağıdaki hata mesajı beni bekliyordu.



Vouv! Hiç beklemediğim bir durum. 7nci eleman yani �a� değerinin karakter karşılığı oluvermişti bir anda. Hımm. Aslında bu istemediğim bir sonuç. Ben daha çok bir hata mesajı bekliyordum. Yoksa dizi tanımında bir tanımlama hatasımı yaptım? Denemeye devam etmeye karar verdim. Sabahın kaçı olursa olsun ben bu hata mesajını bulacaktım. Kodları aşağıdaki şekilde değiştirdim. Aklıma ilk gelen integer değilse bir double değeri buraya koymak oldu. Hatta bir tane değil bir kaç farklı veri tipinden değerler koymak.

```
public class Dizi2
{
    public static void main(String[] args)
    {
        int[] dizi={1,"Selam naber",5,true,9,11,15.154654,15};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
```

```
        }
    }
}
```

Sonunda başardım. İstediğim hata mesajlarını aldım ve dizi tanımına sadık kaldım.

```
C:\JavaSamples\diziler>javac Dizi2.java
Dizi2.java:6: possible loss of precision
found    : double
required: int
                int[] dizi={1,3,5,7,9,11,15.154654,15};
                                         ^
1 error

C:\JavaSamples\diziler>javac Dizi2.java
Dizi2.java:6: incompatible types
found    : java.lang.String
required: int
                int[] dizi={1,"Selam naber",5,true,9,11,15.154654,15};
                              ^
Dizi2.java:6: incompatible types
found    : boolean
required: int
                int[] dizi={1,"Selam naber",5,true,9,11,15.154654,15};
                                              ^
Dizi2.java:6: possible loss of precision
found    : double
required: int
                int[] dizi={1,"Selam naber",5,true,9,11,15.154654,15};
                                                        ^
3 errors

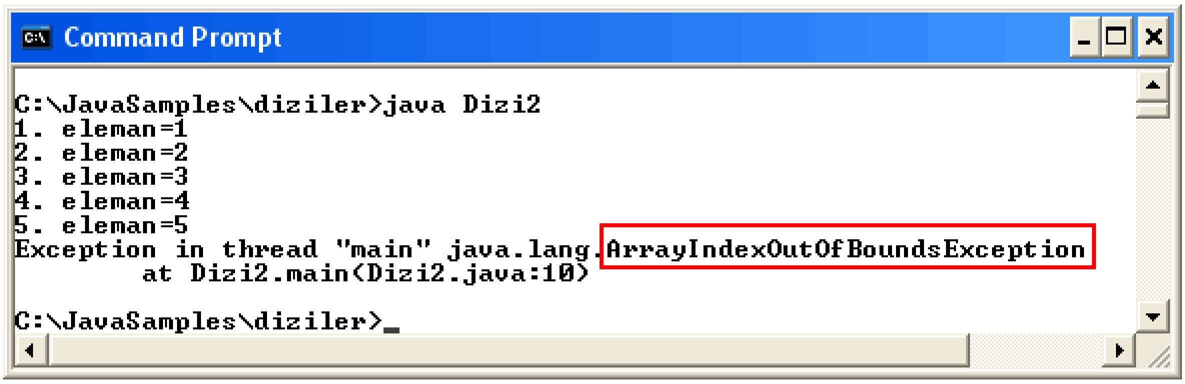
C:\JavaSamples\diziler>_
```

Ancak dizileri kullanırken başıma gelebilecek diğer hatalarıda incelemek istiyorum. Öncelikle dizide olmayan bir indis elemanı oluşturmak istersem veya ona ulaşmak istersem ne olur? Fazla söze gerek yok hemen kodları yazmak lazım. Bu kez 5 elemanlı bir diziye 6ncı elemanı eklemeye çalışacağım.

```
public class Dizi2
{
    public static void main(String[] args)
    {
        int[] dizi={1,2,3,4,5};
        for(int i=0;i<dizi.length;++i)
```

```
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        dizi[6]=7;
    }
}
```

Uygulama yine hatası olarak derlendi. Umarım bu sefer çalıştırdığımda istediğim hatayı alabilirim. Evet. Mükemmel. Hatayı elde etmeyi başardım.



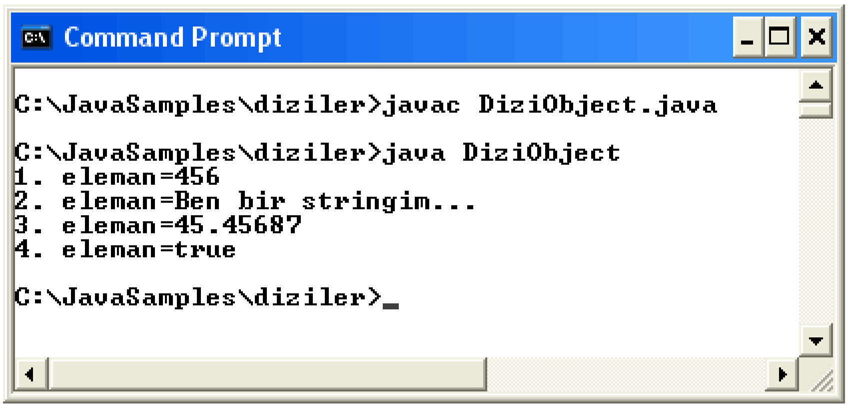
Dizileri tanımlarken, aynı veri türünden elemanları grupladığımızdan bahsetmiştik. Farklı bir durum daha vardır. Bu da bir diziyi Java dilinin en üst sınıf olan (C# taki gibi) Object türünden tanımlamak ve elemanları birer nesne olarak belirlemektir. Bu teknikte, dizinin tüm elemanları birer nesnedir. Bu anlamda bellekteki dizilişte farklı olmaktadır. Öyleki, Object türünden bir dizi tanıladığımızda, dizinin elemanları öbekte tutulurken her bir eleman nesne olduğu için, yine öbekteki asıl alanlarına referans eden değerleri taşırlar. Ancak dizi elemanlarına eriştiğimizde, bu nesnelerin referans ettikleri öbek alanlarındaki verilere ulaşılır. Hemen bununla ilgili bir örnek yapmakta fayda var.

```
public class DiziObject
{
    public static void main(String[] args)
    {
        Integer eleman1=new Integer(456);
        String eleman2=new String("Ben bir stringim...");
        Double eleman3=new Double(45.45687);
```

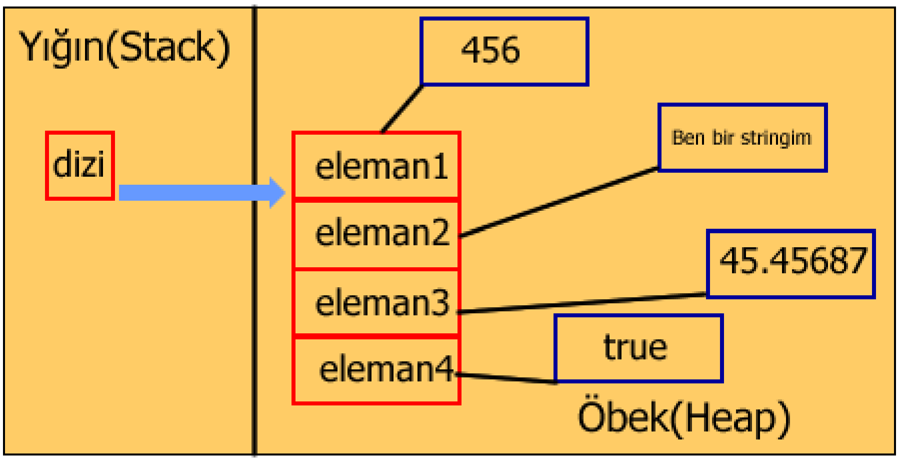
```
        Boolean eleman4=new Boolean(true);

        Object[] dizi={eleman1,eleman2,eleman3,eleman4};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```

Burada yaptığım Object türünden dizim için, new yapılandırıcılarını kullanarak sırasıyla integer, string, double ve boolean veri türlerinden birer nesne yaratmak ve daha sonra bu nesneleri Object türünden diziye aktarmak. Uygulama başarılı bir şekilde derlenicektir ve çalışacaktır.

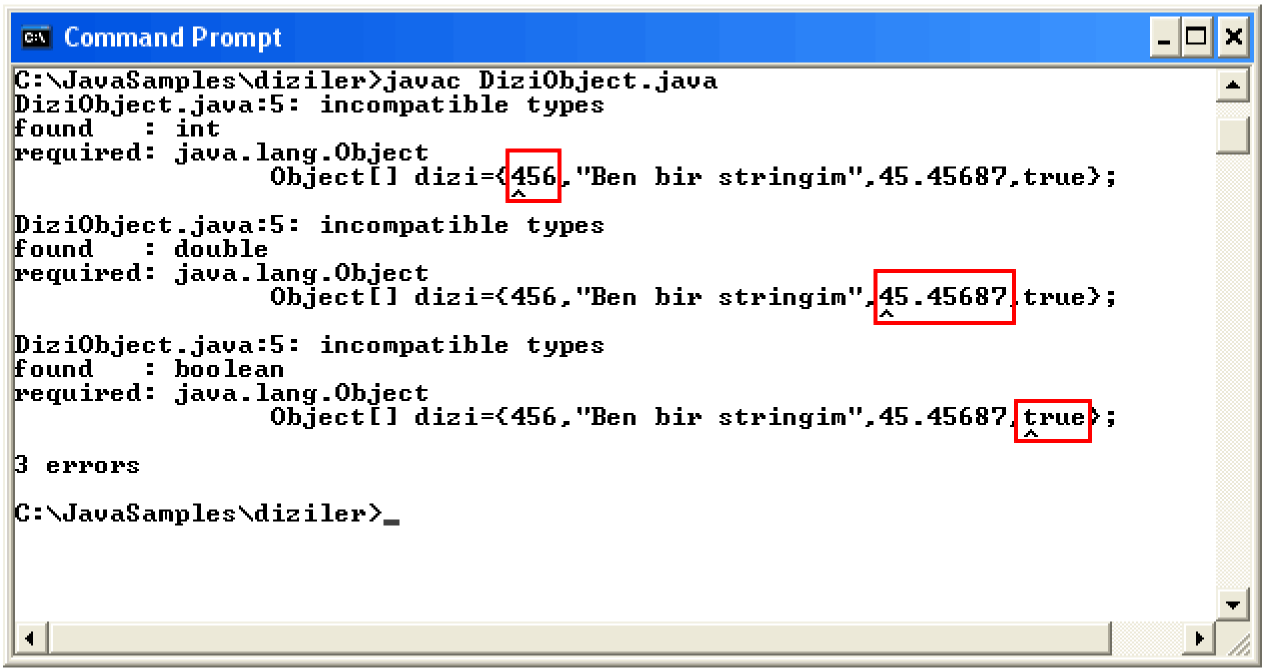


Bu tekniğin bellekteki görünümü ise aşağıdaki tasvire benzer şekilde olucaktır.



Tabi burada aynı örneği aşağıdaki şekide geliştirmeyide düşündüm ama bu durumda hata mesajları ile karşılaştım. Çünkü aşağıdaki tekniği uygulayarak bir Object dizisi oluşturmaya çalıştığımda, elemanların her biri birer değişken olarak algılandı ve Object dizisine eklenmeye çalışılmak istendi. Ancak Object türünden bir dizi, kendisi gibi nesne elemanlarını kabul edeceğinden temel veri türlerinden tanımlanmış değişlenleri kabul etmedi ve derleyici bir sürü hata mesajı verdi.

```
public class DiziObject
{
    public static void main(String[] args)
    {
        Object[] dizi={456,"Ben bir stringim",45.45687,true};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```



Ancak burada dikkat etmemiz gereken bir nokta var. O da, String tipteki "Ben bir stringim" için bir hata mesajı almamış olmamız. Bunun nedeni aslında çok açık. Nitekim String veri türü, bir referans türüdür. Dolayısıyla new yapılandırıcısı ile bir

String nesnesi yaratabiliriz. Bununla birlikte new yapılandırıcısın kullanmadan yukarıdaki gibi bir string tanımlamakta yine aynı işlevi gerçekleştirecektir. Sanırım burada bir dip not olarak şu iki ifadeninde eşit olduklarını söyliyebiliriz.

String s=new String("Merhabar");	String s="Merhaba";

Gelelim diziler ile ilgili ne tür işlevleri gerçekleştirebileceğimize. Dizilerin Java�nın bir sınıfı olduğu söylemiştik. Evet, diziler aslında, java.util.Arrays sınıfının örneklenmiş nesneleridir. Geçen kahve molamdada internetten indirdiğim jsdk dökümanları içinde bu sınıfı aradım ve sayısız pek çok metodunun olduğunu gördüm. Bunların hepsini incelemek için sanıyorumki bir kaç hektar kahve tarlamın olması, bunları öğütücek ve işlemden geçirerek sütlüsünü, 5 şekerlisini çıkartıp bana sunacak bir fabrikam olması gerekiyor. Ama sanıyorumki bu kahve molasında en çok işime yarayacakları incelesem iyi olur. İlk gözüme çarpan aynı isimli metodlardan hemen her veri türü için aşırı yüklenmiş varyasyonların oluşuydu. Örneğin sort metodunu ele alalım. Adındanda anlaşılacağı üzere bu metod dizi elemanlarını sıralıyor. Elbette dizinin tipine bağlı olarak gerçekleştirilen bir sıralama bu. Aşağıdaki örnekte, integer değerlerden oluşan bir diziyi şu metod prototipini kullanarak sıralattım.

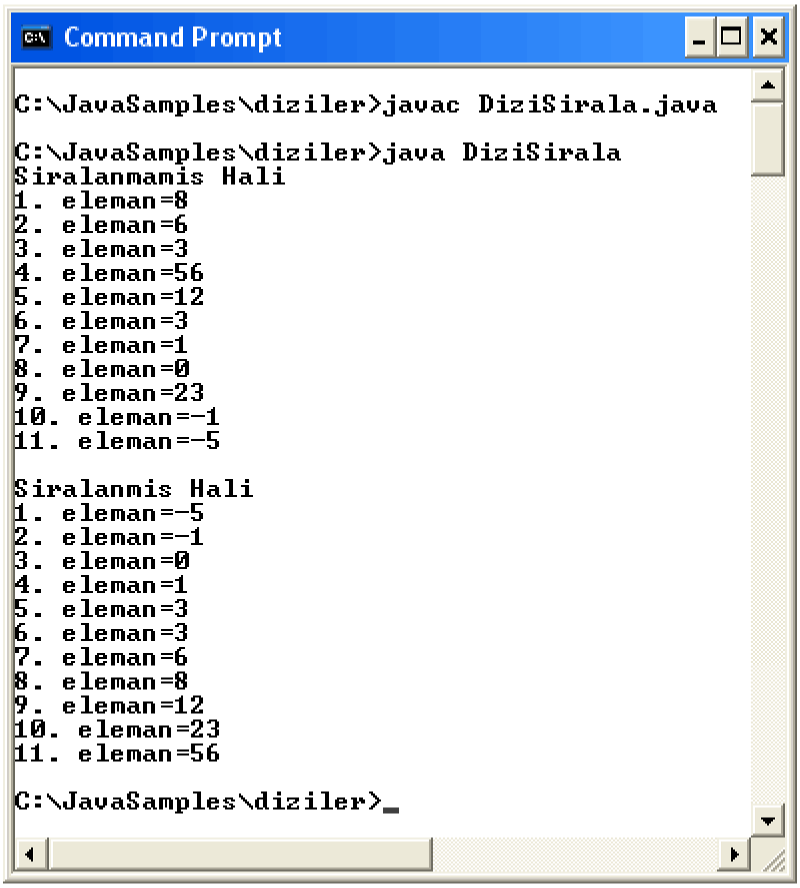
```
public static void sort(int[] a)
```

Şimdide kodlarımızı yazalım.

```
import java.util.*;

public class DiziSirala
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        System.out.println("Siralanmamis Hali");
        for(int i=0;i<dizi.length;++i)
        {
```

```
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        System.out.println("");
        System.out.println("Siralanmis Hali");
        Arrays.sort(dizi);
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```



Burada öncelikle Arrays.sort metodunu kullanarak sıralam işlemini yapmak için, java.util paketini import anahtar kelimesi ile uygulamamıza ekledik. Bu C# taki namespace�lerin using ifadesi ile eklenmesi ile aynı şey. Integer veri türleri için tasarlanmış bu metodun diğer veri türleri için aşırı yüklenmiş versiyonlarıda mevcut. Burada metodumuz parametre olarak dizinin adını alıyor ve dizinin tipine bakarak uygun olan sort

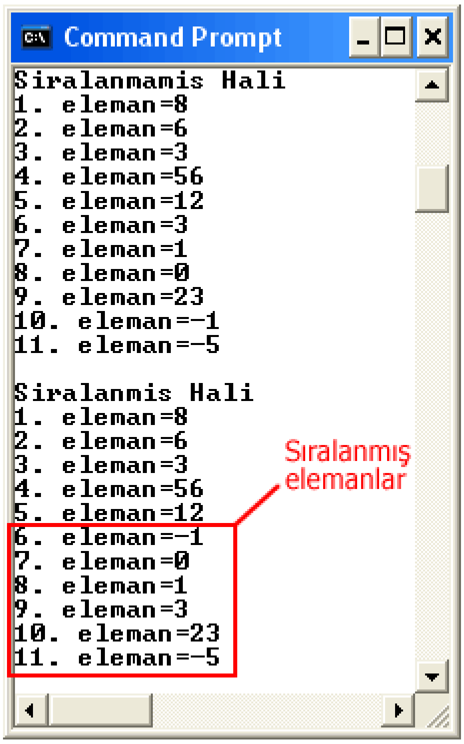
metodunu çalıştırıyor. Sort metodunun bir diğer versiyonuda aşağıda prototipi verilen versiyondur.

```
public static void sort(int[] a,int fromIndex,int toIndex)
```

Bu aşırı yüklenmiş versiyona göre, dizinin sadece belirli indisler arasındaki elemanlarının sıralanmasını sağlamış oluyoruz.Bunu yukarıdaki örneğimize uygularsak;

```
import java.util.*;

public class DiziSirala
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        System.out.println("Siralanmamis Hali");
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        System.out.println("");
        System.out.println("Siralanmis Hali");
        Arrays.sort(dizi,5,10);
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
    }
}
```



Bir diğer ilginç metod ise BinarySearch metodu. Sort metodunda olduğu gibi bu metodda diğer tüm veri türleri için aşırı yüklemiş. Prototipi aşağıdaki gibi olan bu metod, dizi içindeki bir elemanın var olup olmadığına bakıyor ve buna göre bir sonuç döndürüyor.

```
public static int binarySearch(int[] a,int key)
```

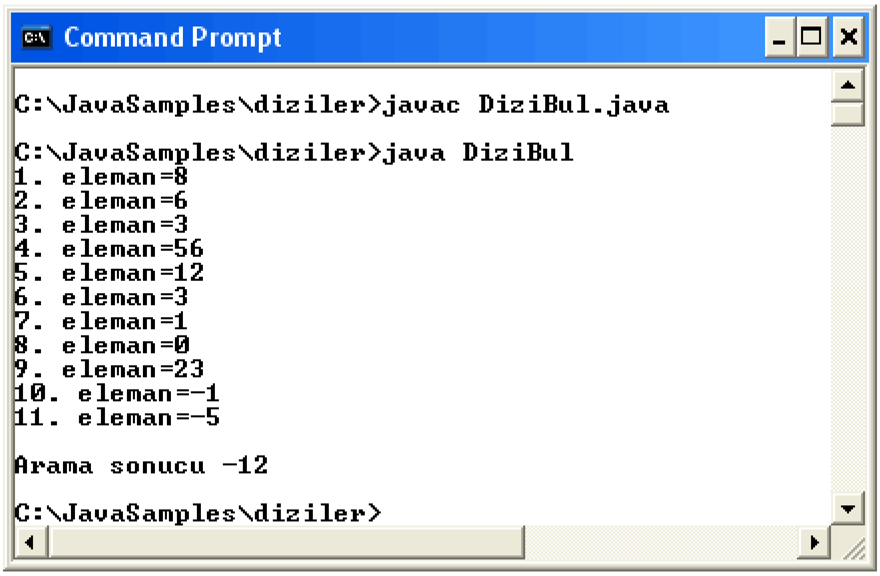
Şimdi bu metod ile ilgili bir deneme yapmanın tam sırası.

```
import java.util.*;

public class DiziBul
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        System.out.println("");
        int sonuc=Arrays.binarySearch(dizi,56);
        System.out.println("Arama sonucu "+sonuc);
```

```
    }
}
```

İlk sonuçlar hiçte iç açıcı olmadı benim için. Herşeyden önce aşağıdaki gibi anlamsız bir sonuç elde ettim.

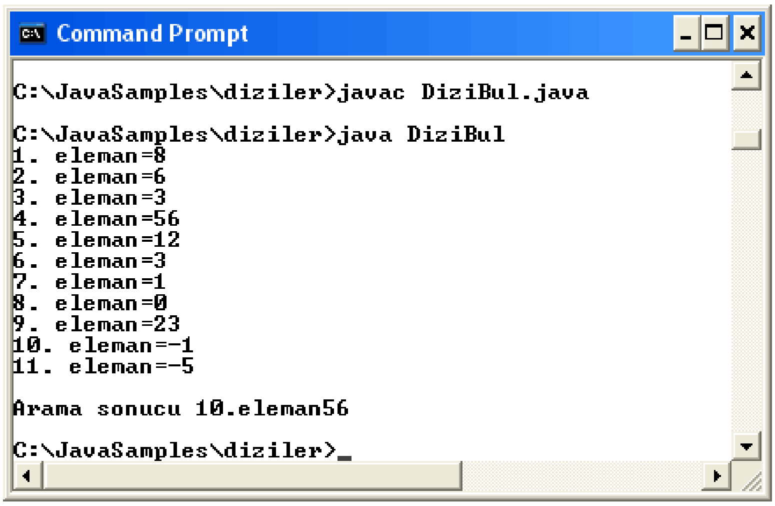


Dökümantasyonu yeniden incelediğimde herşeyin sebebi anlaşılıyordu. BinarySearch metodunu kullanabilmem için, dizinin mutlaka sıralanmış olması gerekiyor. Bu nedenle koda, sort metodunuda ekledim. Bu durumda, dizimizin sıralanmış hali üzerinden arama işlemi başarılı bir şekilde gerçekleştirilmiş oldu. BinarySearch metodunun çalıştırılması sonucu dönen değer, aradığımız elemanın dizi içindeki indis numarasıdır.

```
import java.util.*;

public class DiziBul
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        System.out.println("");
        Arrays.sort(dizi);
```

```
        int sonuc=Arrays.binarySearch(dizi,56);
        System.out.println("Arama sonucu
"+sonuc+".eleman"+dizi[sonuc]);
    }
}
```

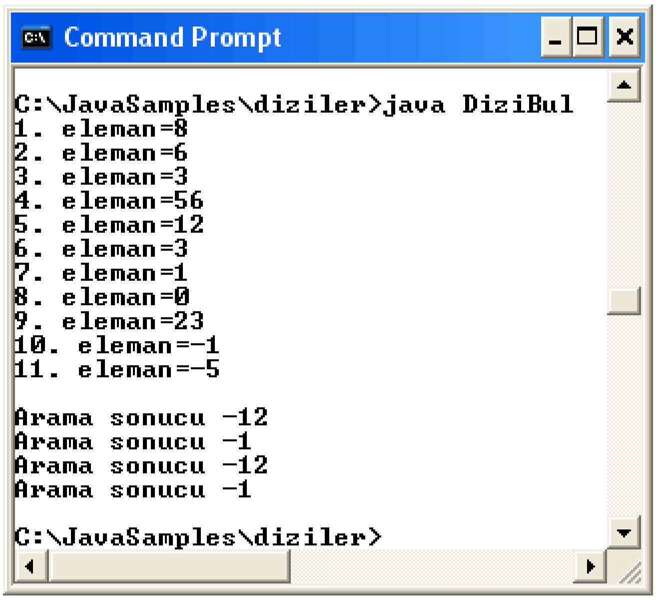


Şimdi daha değişik bir durumu inceleyelim. Eğer dizi içinde olmayan bir eleman ararsak binarySearch metodu nasıl bir sonuç döndürecektir. Bu amaçla aşağıdaki kodu hazırladım. Örnek içinde, iki adet negatif ve iki adet pozitif tamsayıyı dizi içerisinde arattım. Bu elemanların hiçbirisin dizinin bir elemanı değil. Sonuçlara baktığımda dizide olmayan pozifit elemanlar için -12 değerini, dizide olmayan negatif elemanlar için ise -1 değerini elde ettim. Pozifit ve negatif elemanlar için binarySearch metodunun döndürdüğü değerler farklı olmasına rağmen ikise negatiftir. Dolayısıyla negatif olmaları dizinin elemanı olmadıklarını göstermektedir.

```
import java.util.*;

public class DiziBul
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        for(int i=0;i<dizi.length;++i)
        {
            System.out.println((i+1)+". eleman="+dizi[i]);
```

```
        }
        System.out.println("");
        Arrays.sort(dizi);
        int sonuc1=Arrays.binarySearch(dizi,156);
        System.out.println("Arama sonucu "+sonuc1);
        int sonuc2=Arrays.binarySearch(dizi,-8);
        System.out.println("Arama sonucu "+sonuc2);
        int sonuc3=Arrays.binarySearch(dizi,1000);
        System.out.println("Arama sonucu "+sonuc3);
        int sonuc4=Arrays.binarySearch(dizi,-4568);
        System.out.println("Arama sonucu "+sonuc4);
    }
}
```

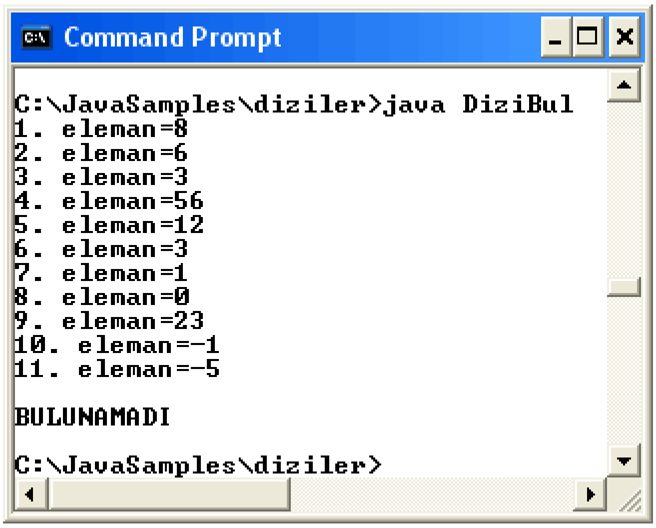


Demekki binarySearch metodunu bir if koşulu ile kullanarak aranan elemanın dizi içerisinde olup olmadığını belirleyebiliriz.

```
import java.util.*;

public class DiziBul
{
    public static void main(String[] args)
    {
        int[] dizi={8,6,3,56,12,3,1,0,23,-1,-5};
        for(int i=0;i<dizi.length;++i)
        {
```

```
            System.out.println((i+1)+". eleman="+dizi[i]);
        }
        System.out.println("");
        Arrays.sort(dizi);
        int sonuc1=Arrays.binarySearch(dizi,156);
        if(sonuc1<0)
        {
            System.out.println("BULUNAMADI");
        }
        else
        {
            System.out.println("BULUNDU");
        }
    }
}
```



Arrays sınıfı ile ilgili daha pek çok metod vardı ve bunlarıda incelemek istiyordum. Ama hava aydınlanmaya ve içmeyi unuttuğum kahvemde ice-cafe olmaya başlamıştı. Sanırım ara vermek için en uygun zaman. Biraz dinlenmeyi hakkettik. İlerleyen kahve molalarında kahvemi içmeyi unutmayacağım.

Bölüm 4: Kahvenin Tadı Sınıflarda Saklı

Pazar sabahları ne yaparsınız? Peki bir bilgisayar programcısı olsanız ne yaparsınız? Muhtemelen, cumartesi gecesinin sabaha kadar süren çalışmalarından başınızı kaldıramadığınız için, bütün gün uyumak isteyebilirsiniz. Ben çoğunlukla ve düzenli olarak, İstanbul Bostancı sahilinde çok erken saatlerde yürüyüş

yaparım. Sabah 6 ile 7 arası inanılmaz bir huzur bulurum burada. Bir yandan denizden gelen tertemiz iyotlu hava, diğer yandan sessizlik, martı seslerinin ahengi ve göz kamaştırıcı güzelliği ile İstanbul boğazının Marmara denizi ile kucaklaşması. Matematikçi olmamın bana verdiği kazanımlardan birisi, etrafta olup bitenleri son derece felsefik inceleyebilme yeteneği. Sizde bilirsiniz ki tarihin en ünlü matematikçileri mantık, sosyoloji, felsefe, psikoloji gibi bilimlerle hep yakından ilgilenmişlerdir. Onlardan birisi olmayı çok isterdim gerçekten.

Herşeyden önce bir programcı pek çok zorlukla, hayatta karşılaştığından çok, programlarını yazarken karşılaşır. O nedenle, sağlıklı ve ruhsal açıdan huzurlu bir beden, onun bu zorlukları aşmasında en büyük yardımcıdır. Ben bu yürüyüşleri çoğunlukla kafamı boşaltmak, kendi dünyamda huzur bulmak ve gözümden kaçan ayrıntıları daha sağlıklı inceleyebilmek amacıyla yaparım. İşte bu sabahta o amaçla yapılan bir yürüyüşteyim.

Adımlarımı attıkça, ciğerlerime dolan nefis iyotlu deniz havası daha sağlıklı düşünmemide sağlıyor. Etrafıma baktığımda, her türden nesneyi daha berrak sezinleyebiliyorum. Nesne! O o! Bu lafı bana kim söyletti? Nesne...Nesne...Gerçekten de şöyle bir durdum ve etrafıma bakındım. Her yerim , ben de dahil olmak üzere nesnelerden oluşmaktaydı. Evrenin sahip olduğu gizemi ve mükemmelliği kimse inkar etmez. Çağımız boyunca ve gelecektede pek çok teknolojik, endüstüriyel, matematik yaklaşıma ilham kaynağı olmuş sistematikler içerir. Bu düşünceler ile tekrar yürümeye başladığımda, nasıl olduysa bir anda kendimi bilgisayar dillerinin gelişimini düşünürken buluverdim. Basic, pascal, C, Fortran vesaire... Sonra tekrar durdum. Aklıma C++,Java,C# gelivermişti birden bire. Hepside nesneler ile uğraşan, nesneler üzerine kurulu yapılara sahipti. Hepside Nesneye Dayalı Programlama dillerindendi.

Nesne kavramını, günümüz modern programlama dillerine kim yerleştirmiş ise eminimki benim gibi bir yürüyüş sırasında bunu yapmıştır diye düşünüyordum. Bir anda kendimi Yazılım Mimarları gibi düşünürken, ayağımı yere sürtüp tepe taklak tabir yerinde ise iki seksen yere uzandım. Allahın tokatı yoktur derler. Neyseki kendime gelmiştim. O adamların tırnağı bile

olmak büyük bir onur olurdu sanırım. Evet sonuç itibariyle sabahları bu kadar erken yürümemin nedeni buydu. Tökezleyip düştüğümde etrafta kimsenin olmayışı...Sonuç itibariyle geriye dönüp baktığımda, hem programlama dillerinin nesneye dayalı hale gelmesi ile, yaşamımızın ekosistemini örnek alarak ne kadar büyük üstünlükler sağladığını düşünüyor, bir yandanda ne kadar uzağa yürümüşüm şimdi nasıl geriye dönecem diye veryansın ediyordum. Aklıma her zaman olduğu gibi parlak bir fikir geldi. Dönüşte çevremdeki nesnelerin programlama dillerindeki etkilerini düşünecektim.

Çevremizde ne kadar çok nesne var. Çiçekler, böcekler, insanlar, arabalar, otobüsler, kuşlar, taşlar, kayalar...Bu liste o kadar kabarıkki saymaya kalkmak bile sonsuzlukla çarpışmak gibi bir şey olsa gerek. Her nesnenin belli bir takım özellikleri, işlevsellikleri ve hatta amaçları var. Bazı nesneler bir araya gelerek başka yeni nesnelerin doğmasına neden olurlarken onlara bir takım ortak özelliklerinide veriyorlar. Aynı insanların bir araya gelerek evlenmesi, çocuk sahibi olması ve çocuklarına kendi özelliklerinden bir takım kalıtımlar bırakması gibi. Bazı nesneler kendi içinde kapalı, sahip olduğu değerleri değiştirilemeyen ama gözlemlenebilen türden. Bazı nesneler ufak değişikliklere uğrayarak başka nesnelere ilham kaynağı olmuşlar. Bu çeşitlilik altındaki tüm neselerin ortak özelliklerini bir arada düşünmek ve bir kenara koymak onları bu ortak niteliklerine göre sınıflandırmaktan başka bir şey değil.

Otomobilleri bir sınıf altında düşünebiliriz. Her otomobil 4 tekerlekli, motoru olan, ileri geri hareket edebilen, direksiyona sahip ve benzinle çalışan, gaza basıldığında viteste ise hareket eden, frene basıldığında duran ve bunlar gibi pek çok ortak özelliğe ve işleve sahip olan birer nesne. Ama bu sınıfa ait tüm nesneler bir birlerinden farklı olabileceği gibi birbirlerinin aynısıda olabilir. Otomobil firmalarının arabalarını düşündüğümüzde, hepsinin farklı özelliklere sahip ama temel işlevsellikleri neredeyse aynı olan nesneler olduklarını söyleyebiliriz. Hatta üretim hattından yeni çıkmış tüm gri renkli opel vectra 1.6'lar motor seri numaraları hariç birbirlerinin aynısı olan nesnelerden oluşan bir nesne koleksiyonundan başka bir şey değildir.

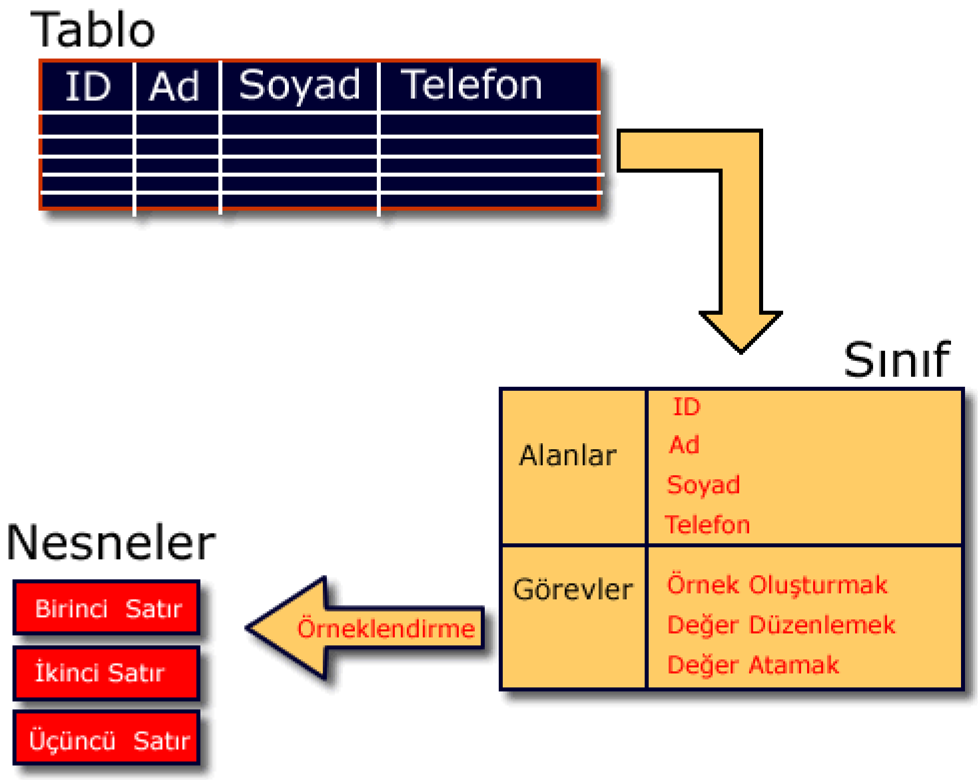
İşte gerçek hayattaki nesnelerin bizler için anlamı ve önemi neyse, nesneye dayalı bir programlama dili içinde nesnelerin anlamı o derece önemlidir. Herşeyden önce, bu dillerde bütün kavramlar, nesnelere ve dolayısıyla bu nesneleri oluşturmak yada örneklemek için kullanılan sınıflara bağlıdır. Düşünün ki object sınıfı değilmidir C# dilinde ve hatta Java'da en üst sınıf. Java programlama dilide tam anlamıyla nesneye dayalı bir dil. Dolayısıyla nesneleri kullanan bir dil. Nesneleri, sahip oldukları özellikleri, değerleri ve işlevleri ile kullanan bir dil. Bu düşünceler eşliğinde geri dönüş yolunu tamamladım. Artık eve dönme vaktim gelmişti. Şu andan sonra yapılacak en güzel şey, eve gitmek sıcak bir duş almak ve Pazar sabahının gazetelerini okuyup güne merhaba demekti. Ama ne yazıkki böyle olmamıştı. İçimdeki korkunç öğrenme açlığı, gazete yerine Java ile ilgili kaynakları araştırmama neden oldu. Ama her zaman olduğu gibi yanımda sıcak bir kahvem vardı.

Uzun süre camdan dışarı bakarak sınıfların yerini düşündüm ve onları daha iyi anlamaya çalıştım. Gerçektende Javada'da diğer nesneye dayalı programlama dillerinde olduğu gibi herşey sınıflar üzerine kurulu idi. Dolayısıyla sınıfları çok iyi kavramak programlama dilini öğrenirken yaşadığım süreçte çok ama çok önemliydi. Önemli olan sadece sınıfların nasıl yazıldığını öğrenmek neleri ihtiva edeceğini bilmek değildi. Sınıfları en uygun şekilde, en etkin ve verimli şekilde kullanabilmekte çok önemliydi. Söz gelimi, bu gün .net dilinde, veritabanlarındaki veriler ile çalışmak için inanılmaz kabiliyetli ve yetenekli sınıflar vardı. Java içinde bunların benzerlerini yazmak istememiz bu tip sınıfları tasarlamamız anlamına geliyordu. Veya veritabanlarından okuduğumuz veri satırlarını düşünelim. Bunları birer sınıf nesnesi olarak uygulamamıza yerleştirmek , işlemlerimizi dahada kolaylaştırmazmıydı. Veritabanına bağlanır istediğimiz satıra ait verileri uygun bir sınıf nesnesine aktarır ve bunları tekrar veritabanına gönderinceye kadar bağlı olmak zorunda kalmazdık. Aynı DataSet kavramı gibi. Ama daha sade, daha kolay ve belkide daha etkili.

Elbette Java dilinde veritabanları ile ilgili ne tür işlemler yapabileceğimi ne tür kabiliyetlere sahip olduğumu şu an için bilmiyorum ama ileride bunları öğrenmek içinde can atıyorum. Ama şu an için yapacağım sınıfların nasıl oluşturulduğunu ve

kullanıldığını anlamak olucak. Java dilini öğrenmeye devam ettikçe sanıyorumki sınıf kavramını çok daha iyi kavrayacağım. Gerçi, bu kavrama C# dilinden oldukça aşine ve hakimim. Ama Sun'ın belkide bilmediğim sürprizleri vardır bu konuda. Neyse deneyip göreceğiz. Normalde kendimce bu kadar çok konuşmam. Aslında daha az söz ve daha çok hareket taraftarıyımdır. Eeee atalırımız ne demiş; nerde hareket orda bereket. Tipik bir uygulamacı işte. O bakımdan makinemin başına geçip ilk sınıfımı yazsam iyi olucak sanırım.

Şimdi bir uygulama geliştirmek istiyorum. Bu uygulama hayali olarak bir tablodan bir kaç satır veri alsın istiyorum ve sonra bu verileri birer nesneye aktarmak istiyorum. Bu amaçla öncelikle kafamda hayali bir tablo tasarladım ve sonrada bu tablonun satırlarını birer sınıf nesnesi olarak nasıl tasarlayabileceğimi düşündüm. Aslında işin tasarım kısmında hep bir şeyler çizmek, düşünceleri kağıda dökmek en güzelidir sanırım. Bunun için aslında çok güzel bir yapı var. UML. Fakat ben buradaki şekilleri ezberlemekten ve unutmaktan bıktım. O nedenle başka birisine şeklimi göstermediğim sürece genelde kendi kafamda oluşturduğum çizelgeleri kullanırım. Sanırım buna ben BML diyeceğim. Diğer yandan çizdiklerimin başkaları tarafından kolayca anlaşılması içinde dikkat ediyorum.



Aslında yapmak istediğim şey çok basit. Tablodaki satırları alıp, Sınıf nesneleri haline getirmek. Bunu yaparken, tablodaki alanların değerlerini, sınıfım içinde oluşturacağım değişkenlerde saklıyacağım. Bu durumda, sınıfımdan nesneler türettikten sonra, içindeki değişkenlere değer atamamı sağlıyacak işlevlere ihtiyacım olucak. Ayrıca program içinde, bir nesneye ait bu alan değerlerini değiştirmek isteyebilirim. Bunun içinde bir veya bir kaç metoda ihtiyacım olabilir. Artık ne yapmak istediğimi bildiğime göre kodları yazmaya başlamanın zamanı geldi. Ne yazıkki yine sıkıcı Notepad editorümüze döneceğiz. Aslında buna bir çare bulmam lazım. Java programlarını yazabileceğim ücretsiz bir yazılıma ihtiyacım var. Aslında kaynaklarımdan en iyi java editörünün Borland firmasının JBuilder ürünü olduğunu öğrenmiş bulunmaktayım. Üstelik bu ürünün Personel sürmüde ücretsiz olarak dağıtılıyormuş. Sanıyorum bu öğleden sonra vaktimi ayırıp bu sürümü internetten indirmek için kullanabilirim. Ama şimdilik notepad ile devam edeceğim.

Evet öncelikle bana, oluşturacağım nesneleri örneklendireceğim bir sınıf lazım. Zaten Java'yı öğrenmeye başladığımdan beri, mutlaka sınıf tanımlamalarını yapıyoruz, nitekim Java dilide diğer nesne yönelimli diller gibi, nesnelere dolayısıyla sınıflara dayanıyor. O halde ne duruyoruz. İşe sınıfımız oluşturmakla başlayalım.

```
public class SinifKisi
{

}
```

Elbette sınıfı böylesine tanımlamak son derece anlamsız ama birazdan sınıfın içine koyacağımız üyeler ile bayağı bir yol katedeceğiz. Biz oluşturduğumuz bu sınıfta türeteceğimiz nesneler ile, tablomuzdaki satırları temsil edicteksek eğer, tablodaki alanlarıda temsil etmemiz gerekir. İşte bunun için sınıfımız içerisinde, tablodaki alanlara denk gelicek alanlar tanımlayacağız. Bu alanların özel yanı private olarak tanımlanacak olmaları. Bunun sebebi, her zamanki gibi kapsülleme (Encapsulating). Kapsüllleme sayesinde, sınıfımız içinde kullandığımız alanların dışarıdan her hangibir etki ile doğrudan değiştirilemelerini engellemiş oluyoruz. Bu durum

bizim isteğimiz dışında oluşabilecek atamaların önüne geçmemize ve illede bu alanların değerleri değişecekse bizim belirlediğimiz bir çizgide olmasına neden oluyor. Bu da programcılıkta ve özellikle nesne yönelimli programlamada çok büyük bir yere sahip olan Kontrol'ün elimizde olmasını sağlıyor.

Aslında kapsüllemeyi , içtiğimiz kapsül şeklindeki ilaçlardan yola çıkarak daha iyi anlayabiliriz. Çoğu zaman kapsül şeklinde ilaçlar içeriz. Bu ilaçların dışındaki koruyucu kapsül içinde, binlerce belkide onbinlerce zerrecik yer alır. Bu zerreciklerin böyle bir kapsül içine alınmasındaki en büyük neden bir arada tutulmak istenmeleri olarak düşünülebilir. Aynı sınıflarımızda kullanacağımız alanları bir arada tutmamız gibi. Diğer yandan, kapsüllenmelerinin en büyük nedeni, istenilen hedefe kadar gidilecek yol üzerinde (ağız, tükürük bezleri, yemek borusu, mide vesaire...) karşılaşılabilecek ve zerrelerin istenmeyen şekilde vücuda karışmasına neden olucak etkilerden korunmaktır. Amaç ilacın istenen yerde kapsülünün istenen etki ile eriyerek, zerreciklerin serbest kalmasıdır. Elbetteki kapsül ilaç örneği sınıflardaki kapsüllemeyi tam olarak açıklayamaz, ancak çok büyük benzerlikler taşır.

İşte sınıflarımızı yazarken de, bu sınıflar içierisinde tanımlanan ve sınıfa ait nesneler tarafından erişilebilen alanların, bizim programladığımız etkiler dışında değiştirilmesini istemeyiz. Bir alanın sadece okunabilir olmasını veya, bu alandaki değerlerin bizim kontrolümüz altındaki metodlar ile değiştirilmesini isteyebiliriz. İşte bu uyumu sağlamak için, nesneler tarafından kullanılan ve sınıf içinde tanımlanan alanları private olarak belirtiriz. Böylece bu alanlara sadece tanımlanmış oldukları sınıf içerisinden erişilebilecektir. Şimdi uygulamamızda, tablo alanlarını temsil edicek sınıf içi özel alanlarımızı oluşturalım.

```
public class SinifKisi
{
    private int fID;
    private String fAd;
    private String fSoyad;
    private String fTelefon;
```

```
}
```

Şimdi C# ile bu sınıfı yazıyor olsaydım bu alanlar için birer özellik tanımlar ve get ile set bloklarını oluşturudum. Ancak kaynaklarıma baktığımda şaşırtıcı bir şekilde bu tarz ifadelerin, yani get ve set bloğu olan tarzda özelliklerin yer almadığını gördüm. Belki bu gözümden kaçmıştı ama tekrar baktığımda iyice emin oldum. Özellik tanımlamaları yerine, özelliklerinin rollerini üstlenecek olan metodları kendimiz yazıyorduk. Aslında bu tuhafıma gitti. Neyse yapacak bir şey yoktu, yeni bir fincan kahve almaktan başka. İşe koyulma vakti. Herşeyden önce bu alanların değerlerinin değiştirilebilmesini istiyorum aynı zamanda değerlerinin okunabilmesinide. O zaman her biri için birer metod yazmam gerekiyor.

```
public class SinifKisi
{
    private int fID;
    private String fAd;
    private String fSoyad;
    private String fTelefon;

    public int getID()
    {
        return fID;
    }

    public String getAd()
    {
        return fAd;
    }

    public String getSoyad()
    {
        return fSoyad;
    }

    public String getTelefon()
    {
        return fTelefon;
```

```
   }
}
```

Önce alanların değerlerini sınıf örnekleri olan nesneler üzerinden okuyabilmek için, her bir alan için bir metod yazdım. Bu metodların tek yaptığı return anahtar kelimesi ile geriye, private sınıf alanlarının değerlerini göndermek. Tabi burada önemli olan nokta, bu metodların dönüş değerlerinin, private alanların veri tipleri ile aynı olmaları. Bununla birlikte bu metodları, sınıf dışından kullanabilmek için yada bu sınıfımızdan türeteceğimiz nesnelerde kullanabilmek için public belirteci ile tanımlıyoruz. Sıra geldi bu alanların değerlerini değiştirecek metodlara. Herşeyden önce, fID alanımız aslında, tablomuzdaki ID isimli birincil anahtar alanını işaret ediyor. Tablomuzun bu değerinin kullanıcı tarafından değiştirilmesi kesinlikle istenmeyecek bir durum. Ancak diğer yandan bu alana ait değerleri okumak isteyeceğimiz, örneğin where sorgularında kullanmak isteyeceğimiz durumlarda olucak. İşte kapsüllme için bir neden daha.

```
public class SinifKisi
{
   private int fID;
   private String fAd;
   private String fSoyad;
   private String fTelefon;

   public int getID()
   {
      return fID;
   }

   public String getAd()
   {
      return fAd;
   }

   public void setAd(String ad)
   {
      fAd=ad;
```

```
    }

    public String getSoyad()
    {
        return fSoyad;
    }

    public void setSoyad(String soyad)
    {
        fSoyad=soyad;
    }

    public String getTelefon()
    {
        return fTelefon;
    }

    public void setTelefon(String telefon)
    {
        fTelefon=telefon;
    }
}
```

Artık bir yerler gelmeye başladık. Ancak sınıfımızda halen daha eksik üyeler var. Herşeyden önce, bir nesneyi yaratmanın en temel yolu new anahtar kelimesini kullanmak. Bu anahtar kelime, tanımlandığı sınıf için bir yapıcı metod çağırır. Yapıcı metodlar, çoğunlukla sınıf örneği olan nesneleri ilk kullanıma hazırlarken kullanılır. Örneğin, sınıf içinde kullanılan alanlara başlangıç değerlerinin verilmesi ve benzeri işlemler için. Bir sınıf için birden fazla yapılandırıcı tanımlayabiliriz. Bu, yapılandırıcı metodların aşırı yüklenmesi ile mümkün olur. Aşırı yükleme bildiğiniz gibi, bir metodun aynı isim altında farklı işlevler için kullanılmasıdır. Burada aynı isimli bu metodları birbirinden ayıran, kullanılan parametrelerin farklılıklarıdır. C# dilinde olduğu gibi aşırı yüklü metodların ayrı şekilde algılanmaları hem metod parametrelerine hemde metodun dönüş tipine bağlıdır. Oysa Java'da durum farklıdır. Java'da aşırı yüklü metodlar sadece parametreleri ile ayırt edilebilir, metodların dönüş tiplerinin, metodların ayırt edilmesinde bir etkisi yoktur.

Sınıfımızı geliştirmeye devam etmeden önce aşırı yüklü yapılandırıcıların bir sınıf içinde nasıl kullanıldığına dair bir örnek verelim.

```
public class OverloadM
{
   private int deger1;
   private int deger2;

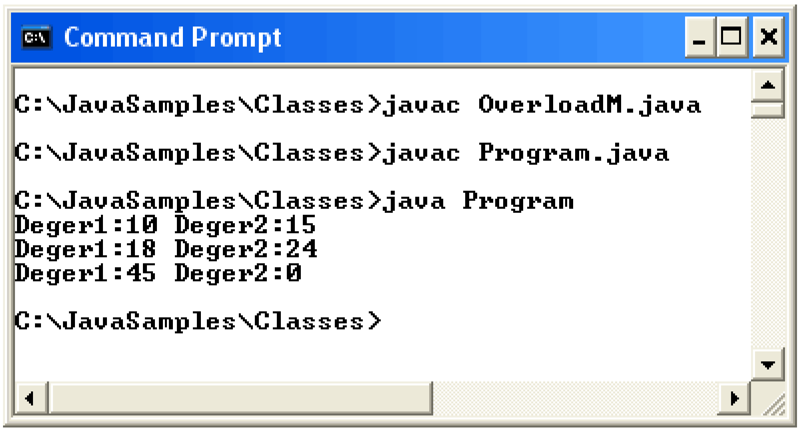
   public OverloadM()
   {
      deger1=10;
      deger2=15;
   }
   public OverloadM(int d1,int d2)
   {
      deger1=d1;
      deger2=d2;
   }
   public OverloadM(int d1)
   {
      deger1=d1;
   }
   public void Yaz()
   {
      System.out.println("Deger1:"+deger1+"
Deger2:"+deger2);
   }
}
```

Burada OverloadM isimli sınıfımızda, üç adet yapılandırıcı metod yer almaktadır. Bunları kullanacak olan sınıfımız ise Program sınıfımız olup kodları aşağıdaki gibi olucaktır.

```
public class Program
{
   public static void main(String[] args)
   {
      OverloadM birinciNesne=new OverloadM();
      birinciNesne.Yaz();
```

```
        OverloadM ikinciNesne=new OverloadM(18,24);
        ikinciNesne.Yaz();
        OverloadM ucuncuNesne=new OverloadM(45);
        ucuncuNesne.Yaz();
    }
}
```

Bu iki sınıfı başarı ile derledikten sonra, Program.java isimli uygulamayı çalıştırdım. Sonuç aşağıdaki gibi oldu.



Görüldüğü gibi yapıcı metodlarımızda dikkat çeken ilk özellik, sınıf adı ile aynı olmaları. Ancak burada karşılaştığım ilginç bir durum oldu. Yapıcı metodların nasıl kullanıldığı ile uğraşırken, varsayılan yapılandırıcıyı tanımlamadığım aşağıdaki örneği yazdım.

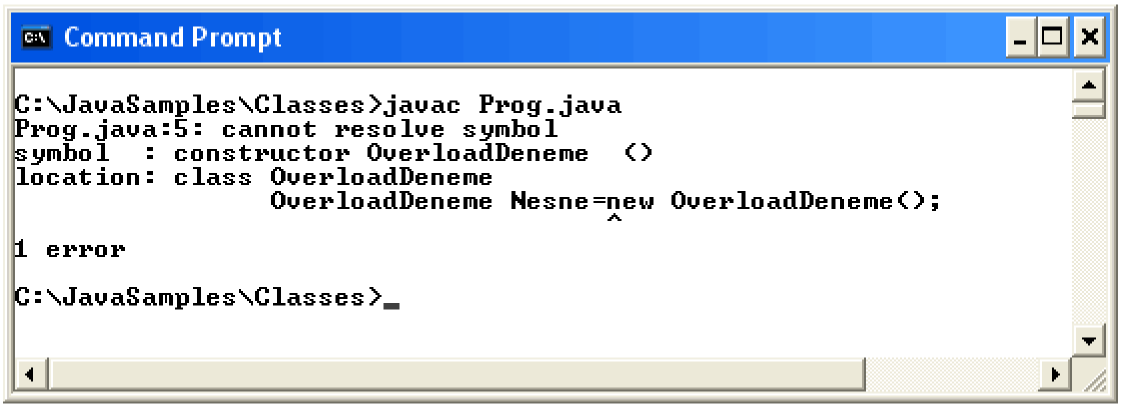
```
public class OverloadDeneme
{
    private int deger1;
    private int deger2;

    public OverloadDeneme(int d1,int d2)
    {
        deger1=d1;
        deger2=d2;
    }
    public OverloadDeneme(int d1)
    {
        deger1=d1;
    }
```

```
    public void Yaz()
    {
        System.out.println("Deger1:"+deger1+"
Deger2:"+deger2);
    }
}

public class Prog
{
    public static void main(String[] args)
    {
        OverloadDeneme Nesne=new OverloadDeneme();
        Nesne.Yaz();
    }
}
```

Prog isimli sınıfımı derlemeye çalıştığımda aşağıdaki hata mesajı ile karşılaştım.



Bir anda ne olmuştuda, new yapılandırıcısı ile basit bir nesne yaratma işlemi hata ile sonuçlanmıştı. Heyecanla ve birazda tedirgin bir havayla, kaynaklarımı hızlı hızlı karıştırmaya başladım. Sonrada Java'nın çok güzel inceliklerinden birini farkettim. Normalde Java, varsayılan yapıcı metodları bir sınıf için yazmassak bizim için bir tane hazırlıyordu. Ancak özelleştirilmiş (parametreler alan) yapıcı metodlar tanımladığımızda, bizim tam anlamıyla konunun hakimi olduğumuzu düşünerek bu varsayılan yapılandırıcıyı otomatik olarak tanımlama desteğini bir anda çekiveriyordu. Tabir yerinde ise bizimle bu konuda muhatap bile olmuyordu. Dolayısıyla varsayılan yapıcı metoduda bizim tanımlamamız

gerekiyordu. Bu güzel incelikten yola çıkarak uygulamayı düzelltim ve hata mesajını ortadan kaldırdım.

C# dilini öğrenmeye ilk başladığım zamanlar üniversite yıllarında öğrettikleri C++ dilindeki büyük küçük harf ayrımında her zamanki gibi ilk başlarda çok unutkanlıklar yaşamış ve hep hatalar yapmıştım. Ama şimdi Java'da gördümkü, unutkanlığa asla yer yok hatta tam anlamıyla dile hakim olmak esas.

Neyse yapılandırıcıların bu özellikleri ve işlevlerinden sonra, acaba kendi sınıfımız için özel bir yapılandırıcıya gerek var mı diye düşünmeye başladım. Bu sınıfa ait nesneler, tablo satırlarını gösterecek olduğundan, yapılandırıclara parametre atayıp, alan değerleri ile oynamak son derece anlamsız geldi. Bu nedenle bir yapılandırıcı yazmaktan vazgeçtim. Ha bu durumda, hiç bir yapılandırıcı yazmadığım için, bu işin kontrolünü Java'ya bırakmış oldum. O benim için varsayılan bir yapılandırıcı oluşturacak ve böylece ben uygulamamda new anahtar kelimesi ile rahatça nesneler yaratabileceğim. Diğer yandan, sınıfımız içine bazı metodlarda eklenebilir. Örneğin, tüm alan değerlerini düzenlediğimizi düşünelim. Bunun için bir metod yazabiliriz. Hatta aynı isimli metodu her bir alan için ayrı ayrı uygulayarak, alanların bireysel olarak düzenlenmesinede imkan sağlayabiliriz diye düşünürken bu işler için metodlar(set ile başlatan metodlar) yazdığımı farkettim. Ancak toplu bir düzenleme için güzel bir metod yazılabilirdi. Böylece sınıfa aşağıdaki metodu eklemeye karar verdim. Bu metod sınıf içindeki alanların değerlerini değiştirecek.

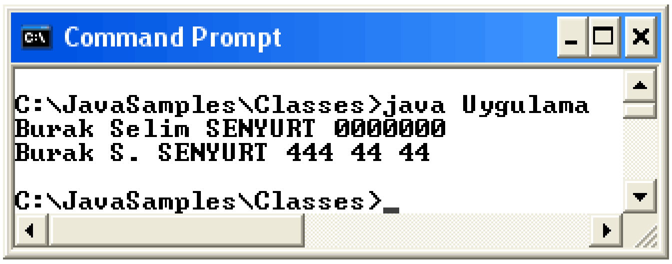
```
public void Degistir(String ad,String soyad,String telefon)
{
   fAd=ad;
   fSoyad=soyad;
   fTelefon=telefon;
}
```

Bu metodun tek yaptığı, parametre olarak aldığı String veri türündeki değerleri alarak, sınıf içindeki alan değerlerine atamak. Sanıyorumki sınıfımın yapısı temel olarak bu şekilde olucak. Artık bu sınıfı ve bu sınıfa ait nesneleri kullanacağım bir örnek uygulamaya ihtiyacım var. Bu uygulamada elbette bir

sınıf şeklinde tasarlanacak ve mutlaka main metodu olucak. Main metodu, bir sınıf için başlangıç noktası niteliğindedir. Ama tabiki tahmin edeceğiniz gibi uygulama içerisinde sadece bir sınıfın main metodu olmalıdır. Bu bilgiler ışığında, aşağıdaki örnek sınıfı hazırladım.

```
public class Uygulama
{
    public static void main(String[] args)
    {
        SinifKisi k1=new SinifKisi();
        k1.setAd("Burak Selim");
        k1.setSoyad("SENYURT");
        k1.setTelefon("0000000");
        System.out.println(k1.getAd()+" "+k1.getSoyad()+"
"+k1.getTelefon());
        k1.Degistir("Burak S.","SENYURT","444 44 44");
        System.out.println(k1.getAd()+" "+k1.getSoyad()+"
"+k1.getTelefon());
    }
}
```

Bu küçük uygulamada, SinifKisi sınıfından k1 isimli bir nesne tanımlıyor ve alanlarının değerlerini değiştirebiliyoruz. Uygulamayı çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Bu noktada merak ettiğim, sınıf içindeki özel alanlara erişmek istediğimde nasıl bir sonuç alacağımdı. O nedenle kodlara aşağıdaki satırları ekledim. Burada açık bir şekilde özel alanlara, tanımlamış olduğum nesne vasıtasıyla erişmeye çalışıyordum. Bakalım derleyici ne yapacak.

```
System.out.println(k1.fAd+" "+k1.fSoyad+" "+k1.fTelefon);
```

Sonuçta uygulamayı derlediğimde aşağıdaki hata mesajları ile karşılaştım. Kapsülleme gerçekleşmişti. Alanları dışarıdaki sınıflardan soyutlamıştım. Onlara sadece benim tanımladığım metodlar vasıtasıyla erişilebilecekti.

```
Command Prompt

C:\JavaSamples\Classes>javac Uygulama.java
Uygulama.java:17: fAd has private access in SinifKisi
                System.out.println(k1.fAd+" "+k1.fSoyad+" "+k1.fTelefon);
                                     ^
Uygulama.java:17: fSoyad has private access in SinifKisi
                System.out.println(k1.fAd+" "+k1.fSoyad+" "+k1.fTelefon);
                                               ^
Uygulama.java:17: fTelefon has private access in SinifKisi
                System.out.println(k1.fAd+" "+k1.fSoyad+" "+k1.fTelefon);
                                                             ^
3 errors

C:\JavaSamples\Classes>
```

Aslında sınıflar ile ilgili daha pek çok kavram vardı kaynaklarımda. Örneğin, static metodlar. Bu metodlar her nesne yönelimli programlama dilinin önemli bir parçası bence. Bir static metoda ne zaman ihtiyaç duyabilirdik?

Static metodlar, tanımlandıkları sınıfın örneklendirilmiş nesnesine ihtiyaç duymadan çalışabilen metodlardır. Dolayısıyla böyle bir metodu kullanabilmek için, bu metodun tanımlandığı sınıfa ait bir nesne yaratmak zorunda değiliz. Doğrudan bu metodun tanımlandığı sınıfa ve pakete nokta notasyonlarını uygulayarak, bu static metodu çalıştırabiliriz. Böylece sistem kaynaklarını özellikle gereksiz nesnelerle doldurmamış oluruz. Örneğin matematiksel işlemlerin yapıldığı metodları göz önüne alalım. Bir sayı dizisindeki elemanları sıralayan bir metod varsayalım. Bu metod mutlaka ve illaki bir paket içindeki sınıf içinde yer alacaktır. Çünkü nesneye dayalı programlama dillerinde herşey nesne modeli üzerine kuruludur ve tüm üyeler sınıflar içerisinde, sınıflarda belli paketler içerisinde yer alır. Paketler ile ilgilide pek çok kaynak buldum. Açıkçası bir kahve molasındada onlar ile ilgileneceğim. Java'daki paketlerin yerleşimi aslında kitaplardaki anlatımlar tarzıyla biraz gözümü korkuttu diyebilirim. Ne demiştik bir sayı disizini sıralayan metodumuz olsun. Bu metodun tek yapacağı iş, parametre olarak aldığı dizinin elemanlarına bir sırlama algoritması uygulamak ve sonucu bir dizi olarak döndürmektir. Bu işi

yapması için bir nesneyi görevlendirmek sistem kaynaklarında sadece but sıralama metodu için yer açmak anlamına gelir. Oysaki metodumuzu static olarak tanımlarsak, nesne yaratmaya gerek kalmayız. Metodumuzu ise sınıf.metod notasyonu ile rahatlıkla kullanabiliriz.

Bu düşünceler ışığında acaba küçük uygulamama bu tip bir static metodu nasıl uygulayabilirim diye düşünmeye başladım. Aklıma bir metod geldi. Örneğin SinifKisi isimli sınıfa ait nesneleri ekrana düzgün bir şekilde yazdıracak bir metodum olsaydı. Aslında bu metodu SinifKisi içine yerleştiripte çağırabilirim. Ama amaç static metodları anlamak olduğu için başka bir sınıf içine almaya karar verdim. İşte o sınıfın kodları.

```
public class Yazmaca
{
   static void KisiYaz(SinifKisi k)
   {
      System.out.println("Kişi adı "+k.getAd());
      System.out.println("Kişi soyadı "+k.getSoyad());
      System.out.println("Kişi telefonu "+k.getTelefon());
   }
}
```

Bir static metod tanımlama için yapılacak tek iş metodun tanımlanmasına static anahtar kelimesini eklemek. Buradaki metod parametre olarak SinifKisi sınıfı tipinden bir nesne örneği alıyor. Şimdi Uygulama.java dosyamın kodlarını aşağıdaki şekilde yeniledim.

```
public class Uygulama
{
   public static void main(String[] args)
   {
      SinifKisi k1=new SinifKisi();
      k1.setAd("Burak Selim");
      k1.setSoyad("SENYURT");
      k1.setTelefon("444 44 44");

      SinifKisi k2=new SinifKisi();
```

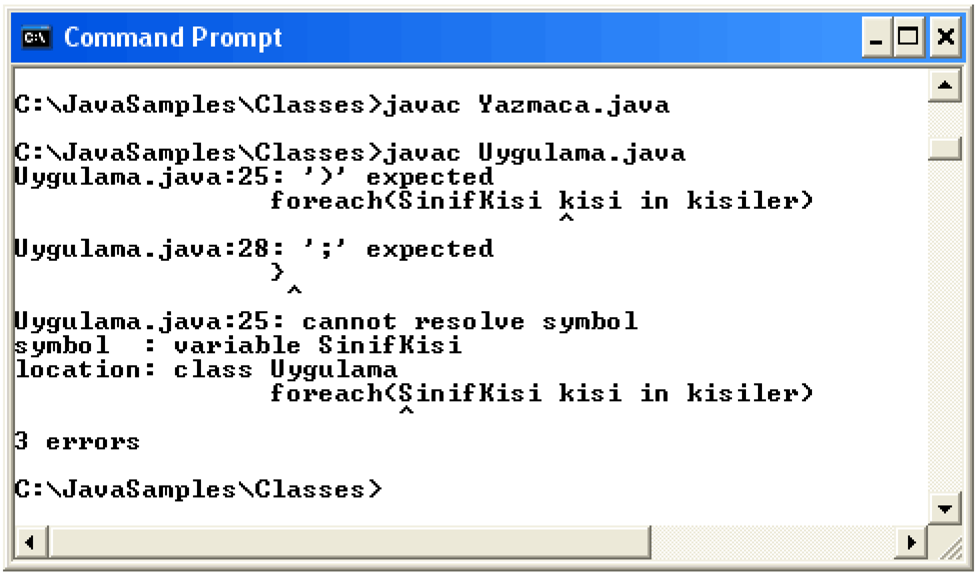
```
        k2.setAd("Sefer");
        k2.setSoyad("ALGAN");
        k2.setTelefon("555 55 55");

        SinifKisi k3=new SinifKisi();
        k3.setAd("Ahmet Faruk");
        k3.setSoyad("NACAROGLU");
        k3.setTelefon("666 66 66");

        SinifKisi[] kisiler=new SinifKisi[3];
        kisiler[0]=k1;
        kisiler[1]=k2;
        kisiler[2]=k3;

        foreach(SinifKisi kisi in kisiler)
        {
            Yazmaca.KisiYaz(kisi);
        }
    }
}
```

Uygulamadaki kodlar bir anda gözüme çok güzel göründü. Üç tane SinifKisi nesnesi yaratıp alanların değerlerini değiştirmiş ve böylece, üç farklı SinifKisi nesnesine sahip olmuştum. Diğer yandan bu nesneleri SinifKisi tipinden bir diziye aktarmış ve bu dizi içindeki her bir elemanı bir foreach döngüsünde ele alarak, static KisiYaz metodunu çağırmıştım. Herşey çok güzel olacak gibi görünüyordu. Ama öyle olmadı. Bam diye bir hata aldım.



Uzunca bir süre düşündüm. Acaba yanlış bir yazımmı uyguladım diye. Ama sonra dökümanlarımda bu işi araştırmaya başladım. Kimse c# taki gibi bir foreach döngüsünden bahsetmiyordu. Acaba böyle bir döngü ifadesi yokmuydu? Ta taaaa.. Hakkatten aradım taradım dökümanlarda bunu bulamadım. Belki başka bir şekli vardı. Bu amaçla elimdeki e-book lardan O'Reilly basımı 6000 sayfalık java dökümanına baktım. Buradada foreach gibi bir döngüden bahsedilmiyordu. Sadece For döngülerini bulabildim. Arayışım bittikten sonra, madem öyle olmuyor bende normal bir for döngüsü kullanırım dedim ve kodları aşağıdaki gibi değiştirdim.

```
public class Uygulama
{
    public static void main(String[] args)
    {
        SinifKisi k1=new SinifKisi();
        k1.setAd("Burak Selim");
        k1.setSoyad("SENYURT");
        k1.setTelefon("444 44 44");

        SinifKisi k2=new SinifKisi();
        k2.setAd("Sefer");
        k2.setSoyad("ALGAN");
        k2.setTelefon("555 55 55");

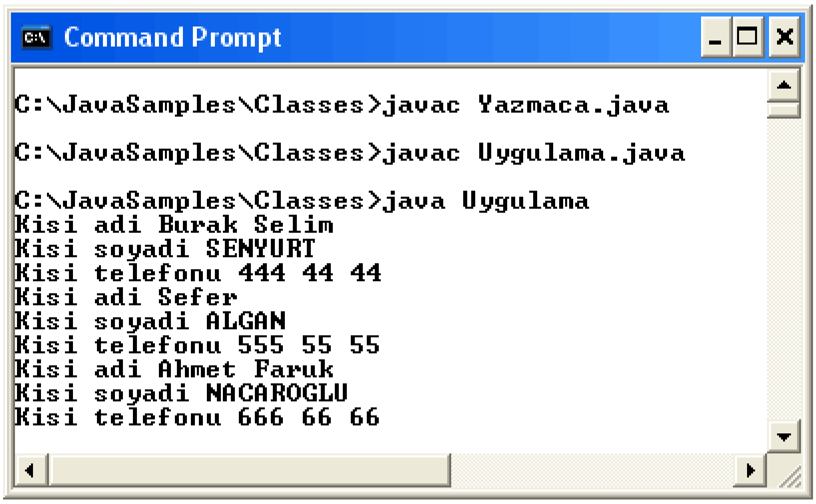
        SinifKisi k3=new SinifKisi();
```

```
        k3.setAd("Ahmet Faruk");
        k3.setSoyad("NACAROGLU");
        k3.setTelefon("666 66 66");

        SinifKisi[] kisiler=new SinifKisi[3];
        kisiler[0]=k1;
        kisiler[1]=k2;
        kisiler[2]=k3;

        for(int i=0;i<=2;++i)
        {
              Yazmaca.KisiYaz(kisiler[i]);
        }
    }
}
```

Şimdi oldu işte. Uygulamayı çalıştırdığımda aşağıdaki sonucu elde ettim. Static metodum olan KisiYaz'a doğrudan sınıf ismi üzerinden erişmeyi başarabilmiştim.



Sınıflar ile ilgili işlenecek daha pek çok kavram vardı aslında. Sınıflar arası kalıtım, çok biçimlilik gibi. Ama kahvemde tükenmişti. Bir süre dinlenmeye ve kaynaklarımı okuyarak yeni bilgiler edinmeye karar verdim. Kim bilir bir sonraki kahve molamda, hangi rüzgarlara kapılacak, hangi memleketlere liman açıcaktım. En azından artık sınıfların Java'da nasıl yazıldıklarını biliyordum. Metodların, özel alanların, static metodların, yapılandırıcıların...Ama katedilecek daha çok yol vardı. Aslında gönlümden geçen, bu uygulamda gerçekten

veritabanına bağlanıp bir tablodan alanları alabilmekti. Fakat yolum çok uzun ve sabretmek lazım. Java'da veritabanlarının işlenmesini çok ama çok merak etmekle birlikte önümde bilmem gereken, Paket kavramı, applet'ler, gui'ler gibi pek çok konu var. Hadi rast gele diyelim.

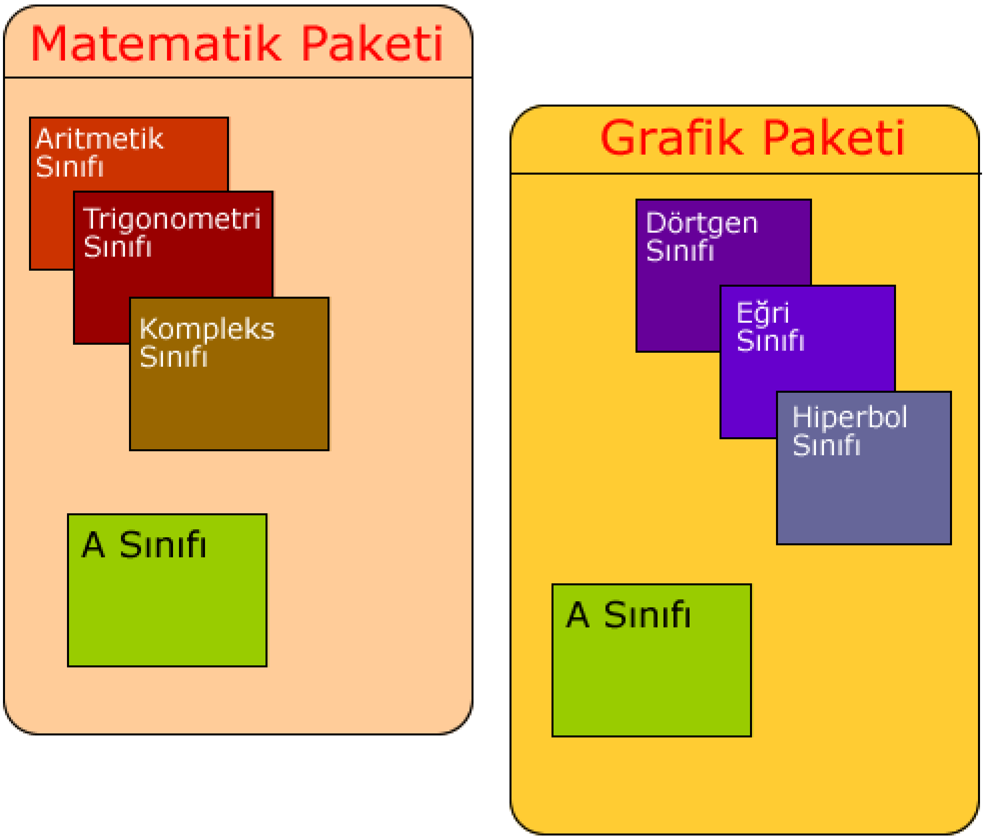
Bölüm 5: Paketler Kargoda

Offf, Kollarım koptu bu paketleri taşımaktan. Bugün, markete kadar yürüyüp bir kaç bir şey alayım dedim eve. Ama gidince habire evdeki bir eksik geliveriyor insanın aklına. Marketlerin insan psikolojisi üzerindeki etkileri işte. Herşeyi almak istiyor insan. Neyse sonunda fazla harcama yapmadan (birazda kendimi tutarak) evime gelebildim. Ama bu paketleri taşımak kollarımı bayağı bir kaslandırdı. Paketleri mutfak tezgahının üstüne bıraktım ve onlara şöyle bir iki saniye duraksayarak baktım. Hafta boyunca, Java�daki paket kavramını incelemeye çalışmıştım. Sanırım, market paketleri ile aralarında bir bağ kurmaya çalışıyordum. Ama sonra kendime geldim ve böylesine bir ruhsal bunalıma girmeye hiç gerek olmadığına karar vermeyi başarabildim.

Sonuçta market paketleri market paketleriydi işte. İşleri bitince çöpe attım gitti. Ya Java paketleri...Hafta boyunca bu konuyu inceledim. Sonuçta elbette buradaki paket kavramı çok tanıdık geldi. Framework için namespace(ad alanı) kavramı ne ise, paket kavramıda Java için oydu. Sonuç olarak, paket kavramı için kafamda şöyle bir tanım oluşturdum. Kısa ve etkili bir tanım. **Paket, içinde sınıf(lar) barındıran bir sistem üyesidir.** Bu tanımdan kısaca paketlerin, içerisinde bir çok sınıf barındırdığı sonucuna varabiliriz. Ama işin önemli kısmı paketlerin bize sağlamış olduğu avantajları iyi bilmekti. İşte bu noktada kaynaklarımdan edindiğim bilgiyi şekle dökmeye karar verdim.

İnanıyorumki bir resim bin kelimeye bedeldir. O yüzden programlama dillerini öğrenirken, bu tarz temel kavramların şekiller ile birlikte açıklanması her zaman için öğrenmemi dahada kolaylaştırmıştı. Örneğin, ado.net içerisindeki sınıfların şekiller ile ifade edilmesi gibi. Bu bakımdan, Java paketlerinin

amaçlarını açıklayabilecek bir şekil üzerinde çalıştım. Burada bir şeklin daha iyi olacağı kanısına varmamda aslında, Yüzüklerin Efendisi üçlemesinin önemli bir payıda var. Günlerden bir gün bir gece vakti, ben askreken, izinim bitmesi nedeni ile Ankara�ya dönüyordum. Yolda gece vakti sessizlikte kitap okumak her zaman hoşuma gider. O günde Yüzüklerin Efendisi filminin ilk bölümünü anlatan kitap elimdeydi. Okumaya başladım ama kısa bir süre sonra bıraktım. Çünkü hobitler, urakhailer, elfler, gandalf filan birbirlerine karışmıştı. Bir baktım ki bir cümleyi iki üç kere okuyorum konuyu anlayabilmek için. Bu böyle olmayacak dedim ve ertesi akşam karargahta görevim bittikten sonra, Kızılay�da bir cep sinemasına giderek filmi izledim. İnanın her şeyi çok daha iyi anlamıştım. İşte bu nedenle konuyu kavramının bazen en iyi yolunun resimler olduğuna inanırım. Sözü bu kadar uzatmanında anlamı yok hani. İşte paket kavramının gerçekleri.



Şekil aslında paket kavramını iyi açıklıyor. Bir paket içerisinde sınıflar olduğunu biliyoruz. Ama ilk amaç aynı konuların amaçlarına hizmet eden sınıfları bir araya toplamaktır. Örneğin

Matematik Paketin�de temel işlemleri barındıran Aritmetik Sınıfı, toplama, çıkarma, çarpma gibi temel aritmetik işlemleri yaparken, Trigonometri sınıfı ise, sin,cos,tan gibi trigonometrik fonksiyonlar üzerinde yoğunlaşır. Aynı şekilde Kompleks sınıfıda, kompleks sayıların işlevleri ile ilgilenir. Diğer yandan sadece çizim işlemleri için kullanabileceğimiz sınıflarıda Grafik Paketi�nde toplayabiliriz. Böylece paketlerin ilk amacına ulaşmış oluruz. **Aynı konulu amaçlar üzerinde yoğunlaşan sınıfları bir çatı altında birleştirmek.**

Diğer yandan, her iki paket içinde A Sınıfı isimli birer sınıf vardır. Bunlar aynı isimli sınıflardır. Ancak farklı paketler içerisinde tanımlandıkları için, bir uygulamada her ikisinide kullanma lüksüne sahibizdir. Tabi burada kullanımda dikkat edilecek bir iki husus vardırki bunları örnekleri yapmaya başladığımda daha iyi anlayacağımızı düşünüyorum. Demekki paketlerin ikinci önemli amacı, **aynı isimli sınıfların sistem içerisinde kullanılabilme imkanına sahip olunması** olarak söyleyebiliriz.

Bir de üçüncü fakat zaten her tanımdanda çıkartılabilen bir amaç vardır. O da **paketleri oluşturarak sınıfların bir çatı altında birleşmesi ve sonradan bu paketlerin kolayca yer değiştirerek farklı platformlara bütün halinde taşınabilmeleri**. Bu aslında oldukça ilginç materyallerin Java�da kullanılmasına neden olmuş bir amaç. Bu amacın sonucu JAR dosyaları oluşmuş. Oldukça gizemli ve güzel bir kavram. Sıkı durun ilerledikçe ondanda bahsedeceğim.

Gelelim Java ile paketlerin nasıl tanımlandığına. Bunun için package anahtar kelimesi kullanılıyor ve paketler oluşturuluyor demeyi çok isterdim. Ancak olay sadece package tanımı içeren bir dosya oluşturmak değil. Bunun dışında paketin isimlendirilmesi ile ilgili güzel bir kural mevcut. İş bu kurala uymaklada kalmıyor sistemdeki CLASSPATH ayarlarınında bu kurala uygun şekilde yapılandırılması gerekiyor. İşte bu bilgiler ışığından hareketle, ilk önce isimlendirme mantığına bir göz atmanın uygun olacağını düşündüm.

Java�da paketleri isimlendirmek için, çoğunlukla internet alan adlarının isimlendirilme mantığı kullanılıyor. Internetteki her alan adı birbirinden bağımsız olarak oluşturulmuştur. Buda onların benzersiz olmalarını sağlamaktadır. İşte bu düşünceden hareket ederek, Java dilinde bir paketi isimlendirmek için, yine bu alan adı mantığı kullanılmaktadır. Hatta Java SDK ile birlikte gelen paketlerde bu isimlendirmeyi kullanır. Örneğin java.awt.image veya org.omg.CORBA gibi. İşte bende ilk paket örneğimi oluştururken bu isimlendirme mantığını kullancağım. Diğer yandan yapılan bu isimlendirmenin bilgisayarımızda fiziki olarak adreslenmesi gerekiyor. Peki bu nasıl mümkün olucak? İşte burada, paket ismimizdeki . ile ayrılan tüm elemanları içiçe klasörlek şeklinde oluşturmamız ve paket dosyamızı en alttaki klasörde yaratmamız gerekiyor. İşte böyle demeyi gene çok isterdim ama yine iş bitmiyor. Çünkü oluşturulan pakete, sistemdeki başka fiziki konumlardanda erişebilmek için CLASSPATH ayarlarını yapmak gerekiyor.

İşte bu bilgileri inceledikten sonra hemen ilk paketimi yazmaya koyuldum. Bu amaçla şimdi yapacağım ve sonradan oluşturacağım paketler için sistemimde ortak bir klasör oluşturdum. Üreteceğim tüm paketleri bu klasör altında toplayacağım.



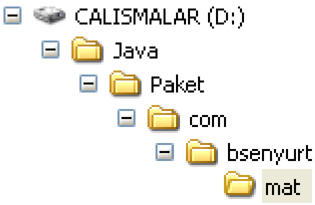
İlk paketim içine, üs alma gibi işlemleri içerecek bir sınıf koymayı düşünüyorum. Bunun gibi matematiksel işlemler ile ilgilenecek bir kaç sınıf daha koyabilirim. Bu amaçla oluşturacağım pakete aşağıdaki adlandırmayı uygulamaya karar verdim.

```
com.bsenyurt.mat
```

Bu arada şunuda not olarak belirtmekte fayda var. Çalıştığım kaynakların tümü, paket isimlendirmelerini yaparken internet alan adı mantığını tersten yazarak uyguluyorlar. Örneğin www.csharpnedir.com adresini baz alarak bir paket oluşturmak

istediğimizi düşünelim. Bu durumda isimlendirmemiz, com.csharpnedir.pkg gibi olabilir.

İlk paketim için oluşturduğum bu isimlendirmenin aynısını, paketleri toplayacağım klasör içerisindede fiziki olarak oluşturmam gerekiyor. Yani klasör yapısı şu şekilde olucak.

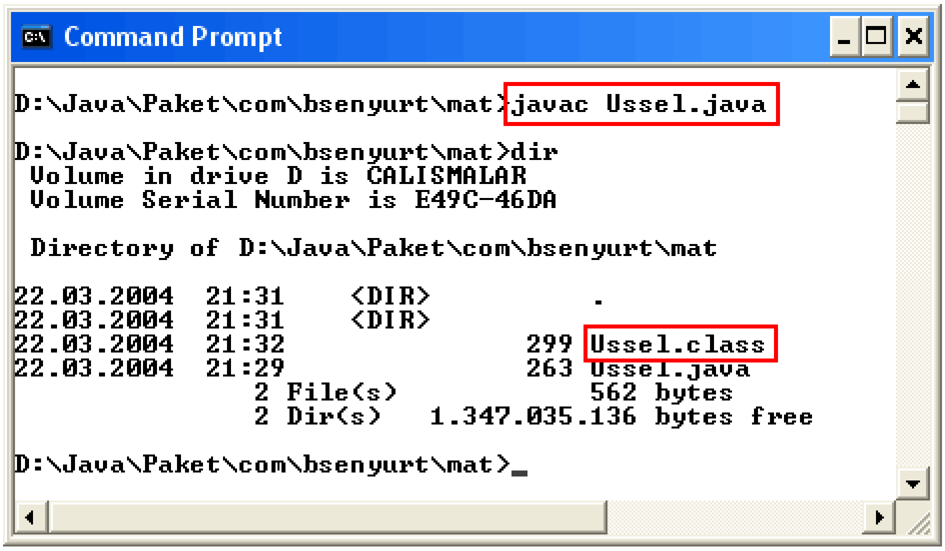


Şimdi paket tanımını içerecek olan java dosyamızı bu klasör içerisine oluşturabiliriz. Bu amaçla, aşağıdaki kod satırlarını içeren java dosyasının oluşturdum. Burada tanımlanmış olan public sınıf adının, dosya adı ile aynı olması gerekmektedir.

```
package com.bsenyurt.mat;

public class Ussel
{
    public double usal(double sayi,double usDegeri)
    {
        double toplam=1;
        for(int i=1;i<=usDegeri;++i)
        {
            toplam=sayi*toplam;
        }
        return toplam;
    }
}
```

Bu kodlarda package tanımını oluşturduğumuz klasör sistemi ile aynı olduğuna dikkat edelim. Ayrıca dosyamızı public sınıfımızın ismi ile kaydediyoruz. Uygulamamızı derlediğimizde, Usler.class dosyasının başarılı bir şekilde oluşturulduğunu görürüz.

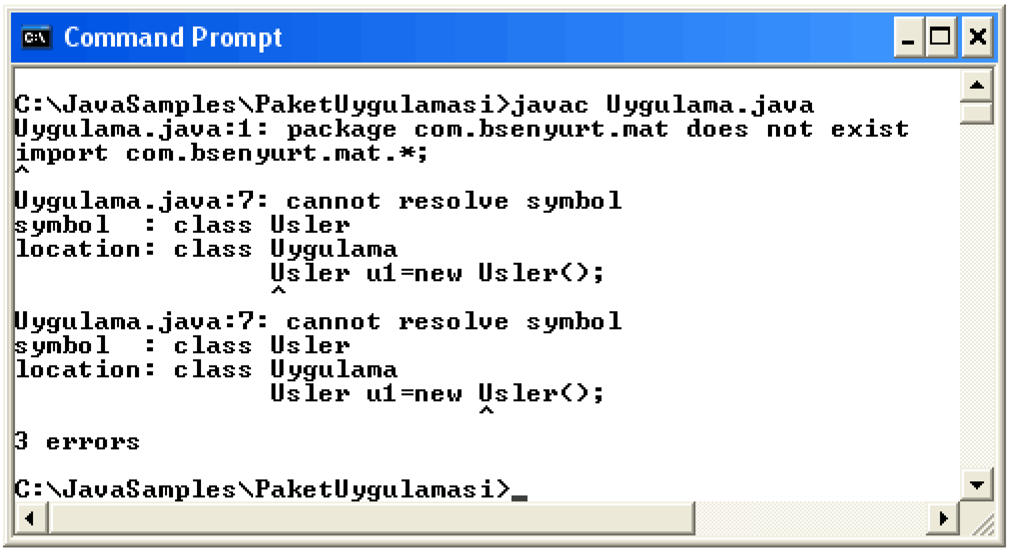


Artık paketimizi oluşturduğumuza göre bunu kullanmanın zamanı geldi. Bunun için paketin bulunduğu klasörden alakasız bir yerde yeni bir uygulama yazmaya karar verdim. Amacım buradan, yazmış olduğum paket içindeki Usler sınıfına ve oradanda usAl metoduna erişmek. Bunun için, sistemde C:\JavaSamples\PaketUygulamasi\ adresinde bir java dosyası oluşturdum. Bu dosyaya aşağıdaki kodları ekledim.

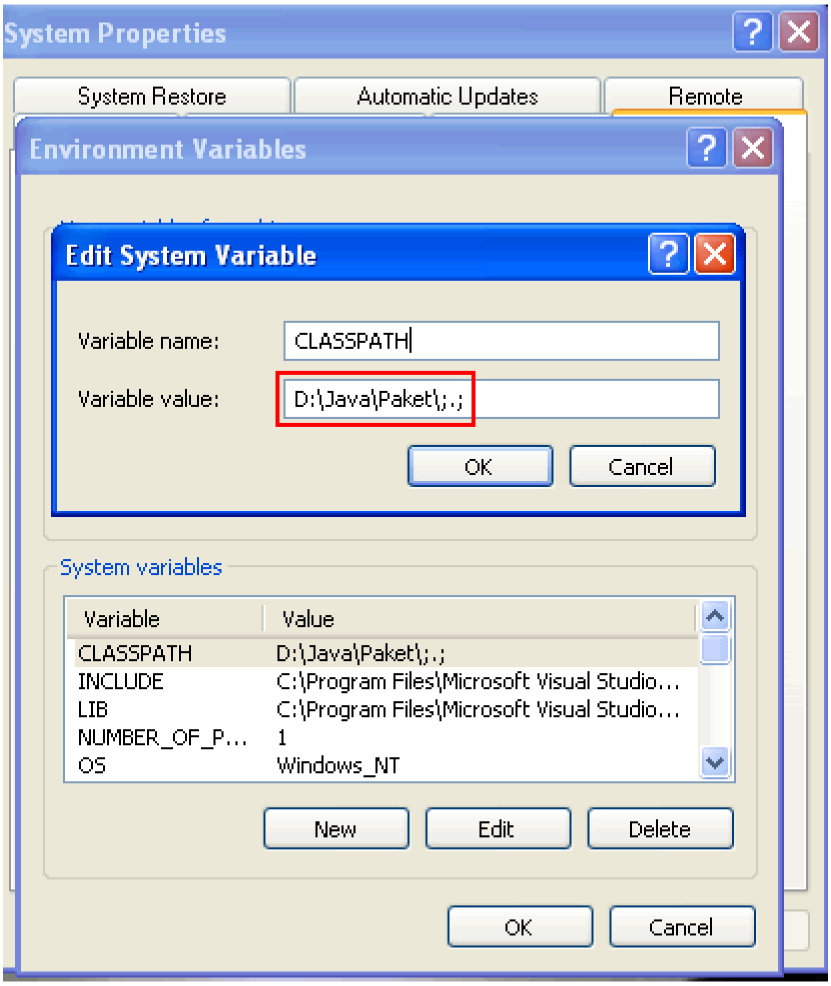
```
import com.bsenyurt.mat.*;

public class Uygulama
{
   public static void main(String[] args)
   {
      Ussel u1=new Ussel();
      double sonuc=u1.usal(2,4);
      System.out.println(sonuc);
   }
}
```

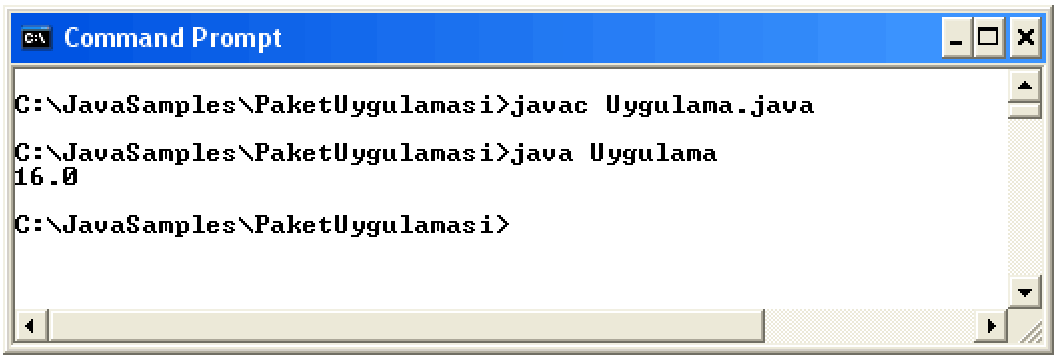
İlk dikkati çeken nokta paketimizin import anahtar sözcüğü ile uygulamamıza eklenmesi. Bu şekilde, bu paket içerisindeki Usler isimli sınıfa erişmeyi umut ediyorum. Ancak bu noktada uygulamayı derlediğimde aşağıdaki hata mesajları ile karşı karşıya kaldım.



Hata mesajlarının anlamı ortaydı. Java derleyicisi, import etmek istediğim paketi bulamıyordu. Java derleyicisinin bu paketi bulabilmesi için, sistemdeki CLASSPATH çevre ayarını değişitirmem ve derleyiciye bu paketin yerini söylemem gerekiyor. Bunun için xp işletim sistemine sahip bilgisayarımda My Computer simgesine sağ tıklayıp Properties�e buradanda Advanced Options kısmına geldim. Daha sonra, Environment Variables kısmında, gerekli CLASSPATH tanımını ekledim. Burada önemli olan com.bsenyurt.mat paketinin içinde bulunduğu klasörün eklenmesiydi.



Böylece Java derleyicisine import anahtar sözcüğü ile eklenen paketi D:\Java\Paket\ klasörü altında araması gerektiğini söylemiş oluyoruz. Şimdi Uygulama.java programını derlersek, uygulamanın başarı ile derlendiğini ve çalıştığını görürüz.



Herşey güzel gidiyor. Ama biz bu pakete başka bir sınıf daha nasıl ekleyebiliriz? Bunun için, oluşturacağımız yeni sınıfı, varolan paketimiz içinde tanımlamalıyız. Aşağıdaki örnekte olduğu gibi.

```
package com.bsenyurt.mat;

public class Temel
{
    public int Toplama(int ilksayi,int ikincisayi)
    {
        return ilksayi+ikincisayi;
    }
}
```

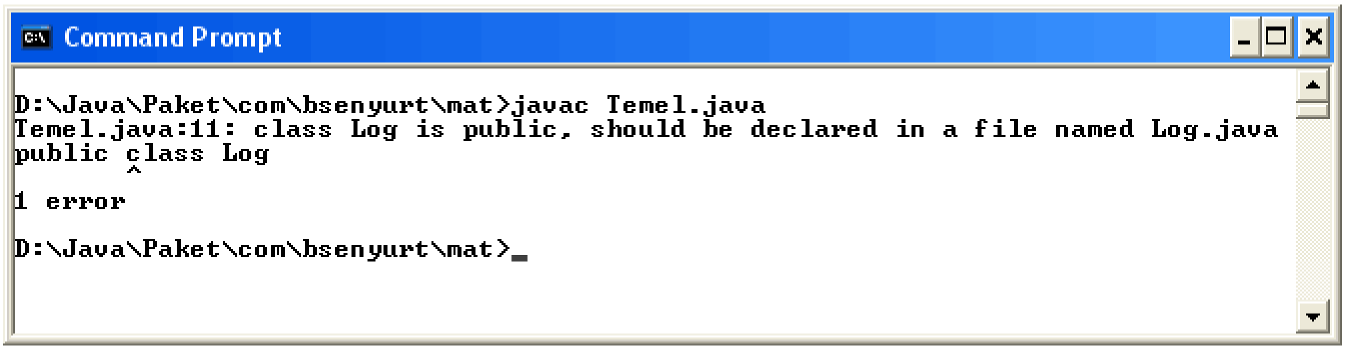
Bu örnek ile, com.bsenyurt.mat isimli pakete Temel isimli sınıfımızıda eklemiş oluyoruz. İlk önce package tanımlamasının yapılması ile, izleyen public sınıfın bu package tanımındaki paket için oluşturulduğunu belirtmiş oluyoruz. Tam bu sırada aklım, C#�taki namespace(ad alanı) kavramına gidiyor. C# �ta bir namespace tanımı altında birden fazla sınıfı belirtebiliyoruz. Acaba Java içinde bu geçerlimi diye merak ediyor ve Temel sınıfın olduğu kodlara başka bir public sınıf ekliyorum.

```
package com.bsenyurt.mat;

public class Temel
{
    public int Toplama(int ilksayi,int ikincisayi)
    {
        return ilksayi+ikincisayi;
    }
}

public class Log
{
    public void LogAl()
    {
        System.out.println("Logaritma alıyor...");
    }
}
```

Ancak programı derlediğimde aşağıdaki hata ile karşılaştım. Public Log sınıfının farklı bir dosyada olması gerektiğine dair bir hata mesajı almıştım.



Fakat bu sınıfı public belirteci olmadan tanımladığımda ise,

```
package com.bsenyurt.mat;

public class Temel
{
    public int Toplama(int ilksayi,int ikincisayi)
    {
        return ilksayi+ikincisayi;
    }
}

class Log
{
    public void LogAl()
    {
        System.out.println("Logaritma alıyor...");
    }
}
```

Uygulamanın başarılı bir şekilde derlendiğini ve Log sınıfı için, otomatik olarak Log.class dosyasının oluşturulduğunu gördüm.

```
Command Prompt

D:\Java\Paket\com\bsenyurt\mat>javac Temel.java

D:\Java\Paket\com\bsenyurt\mat>dir
 Volume in drive D is CALISMALAR
 Volume Serial Number is E49C-46DA

 Directory of D:\Java\Paket\com\bsenyurt\mat

22.03.2004  22:06    <DIR>          .
22.03.2004  22:06    <DIR>          ..
22.03.2004  22:06               414 Log.class
22.03.2004  22:06               261 Temel.class
22.03.2004  22:06               259 Temel.java
22.03.2004  21:57               299 Ussel.class
22.03.2004  21:49               265 Ussel.java
               5 File(s)          1.498 bytes
               2 Dir(s)   1.347.035.136 bytes free

D:\Java\Paket\com\bsenyurt\mat>_
```

Şimdi ise, Uygulama.java dosyasında, tanımlamış olduğum Log.class sınıfına ait bir nesne örneği oluşturmak istiyorum. Bu amaçla Uygulama.java dosyasına aşağıdaki kod satırlarını ekledim.

```
Log l=new Log();
l.LogAl();
```

Şimdi uygulamayı derlediğimde ise, bir dizi hatalar serisi ile karşılaştım.

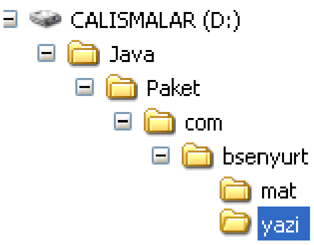
```
Command Prompt

C:\JavaSamples\PaketUygulamasi>javac Uygulama.java
Uygulama.java:15: com.bsenyurt.mat.Log is not public in com.bsenyurt.mat; cannot
 be accessed from outside package
        Log l=new Log();
        ^
Uygulama.java:15: com.bsenyurt.mat.Log is not public in com.bsenyurt.mat; cannot
 be accessed from outside package
        Log l=new Log();
                  ^
Uygulama.java:15: Log() is not public in com.bsenyurt.mat.Log; cannot be accesse
d from outside package
        Log l=new Log();
              ^
Uygulama.java:16: LogAl() in com.bsenyurt.mat.Log is not defined in a public cla
ss or interface; cannot be accessed from outside package
        l.LogAl();
         ^
4 errors

C:\JavaSamples\PaketUygulamasi>
```

Aslında derleme işleminin gerçekleşmeyişinin sebebi gayet açık ve ortadaydı. Eğer bir sınıf tanımına herhangibir erişim belirteci eklemssek (public,private,friendly,protected) bu sınıf friendly olarak kabul edilmekte. Friendly olarak tanımlanmış bir sınıfa ise sadece tanımlanmış olduğu paketten erişebiliriz. Bu nedenle Log sınıfını ayrı bir sınıf dosyası halinde public olarak oluşturmamız gerekir. Aynı Temel sınıfına yaptığımız gibi. Kaynaklarımda erişim belirleyiciler ile ilgili pek çok açıklama var ve bunun başka bir kahve molası olacağını düşünüyorum. Bu nedenle şu anki kahvemi paketler ile birlikte içmeyi tercih ediyorum.

Gelelim paketler arasında aynı isimli sınıfların nasıl yorumlandıklarına. Paketler aynı isimli sınıfları içerebilirler. Peki bu durumda bu sınıfları bir uygulama içinde nasıl kullanabiliriz? Bunu öğrenmek için hemen kolları sıvadım ve path tanımı aşağıdaki gibi olan yeni bir paket tanımladım.




```
package com.bsenyurt.yazi;

public class Temel
{
   public void Uzunluk(String metin)
   {
      System.out.println(metin.length());
   }
}
```

Yazmış olduğumuz bu sınıf, com.bsenyurt.yazi isimli pakete ait bir sınıf. Temel isimli sınıfımız, hem com.bsenyurt.yazi hemde com.bsenyurt.mat paketlerinde tanımlanmış. Gerçi sınıfların içeriği tamamen farklı amaçlara hizmet ettiklerini göstermekte. Şimdi Uygulama.java isimli dosyamıza bu iki paketide import edip, her iki paketten birer Temel sınıfı nesnesi yaratmaya çalışalım.

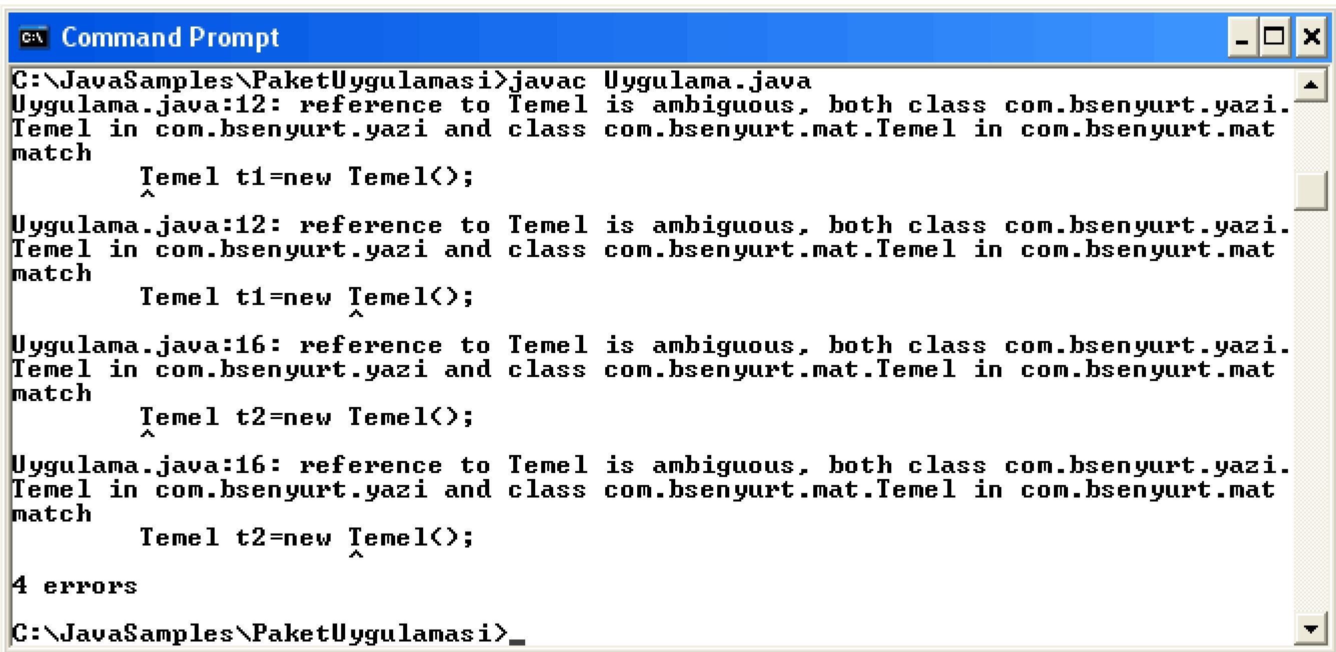
```
import com.bsenyurt.mat.*;
import com.bsenyurt.yazi.*;

public class Uygulama
{
    public static void main(String[] args)
    {
        Ussel u1=new Ussel();
        double sonuc=u1.usal(2,4);
        System.out.println(sonuc);

        Temel t1=new Temel();
        int toplam=t1.Toplama(12,15);
        System.out.println(toplam);

        Temel t2=new Temel();
        t2.Uzunluk("Bu için tükettiğim kaçıncı kahve...");
    }
}
```

Elbette mantık olarak bu uygulamayı daha derlemeden hatalı olduğunu söyleyebiliriz. Hata şudur. Derleyici hangi paketin Temel sınıfını kullanacağını nereden bilecek? Dolayısıyla bu kodları derlemeye çalıştığımızda aşağıdaki hata serileri ile yüz yüze gelmek zorunda kalırız.



Çözüm ise gayet basit. Oluşturulmak istenen nesne sınıfı hangi paketin içerisinde yer alıyorsa, o paket isminin yer aldığı bir notasyonu kullanmak. Aşağıdaki örnek kodlarda olduğu gibi. Örnekte görüldüğü gibi, Temel sınıfının hangi paketteki versiyonunu kullanmak istiyorsak tanımlamaları ona göre yapıyoruz. Dikkat edilmesi gereken bir nokta daha vardır. Kaç paket olursa olsun, ve bu paketler aynı isimde kaç sınıf içerirse içersin, aşağıdaki Temel sınıfına uygulanan teknik, tüm sınıflar için uygulanmalı yani derleyciye bu sınıfın hangi paket içindeki sınıf olduğu açıkça bildirilmelidir.

```
import com.bsenyurt.mat.*;
import com.bsenyurt.yazi.*;

public class Uygulama
{
   public static void main(String[] args)
   {
      Ussel u1=new Ussel();
      double sonuc=u1.usal(2,4);
      System.out.println(sonuc);

      com.bsenyurt.mat.Temel t1=new
com.bsenyurt.mat.Temel();
      int toplam=t1.Toplama(12,15);
      System.out.println(toplam);

      com.bsenyurt.yazi.Temel t2=new
com.bsenyurt.yazi.Temel();
      t2.Uzunluk("Bu için tükettiğim kaçıncı kahve...");
   }
}
```

Son uygulamayıda bitirdikten sonra bir baktım kafamda bir mırıltı. JAR JAR aman JAR ne zaman...diyerek başlayıp giden bir melodiyi mırıldanmaya başladığımı farkettim. Sanıyorumki sonraki yudumumda JAR konusunu işlemem gerekiyor.

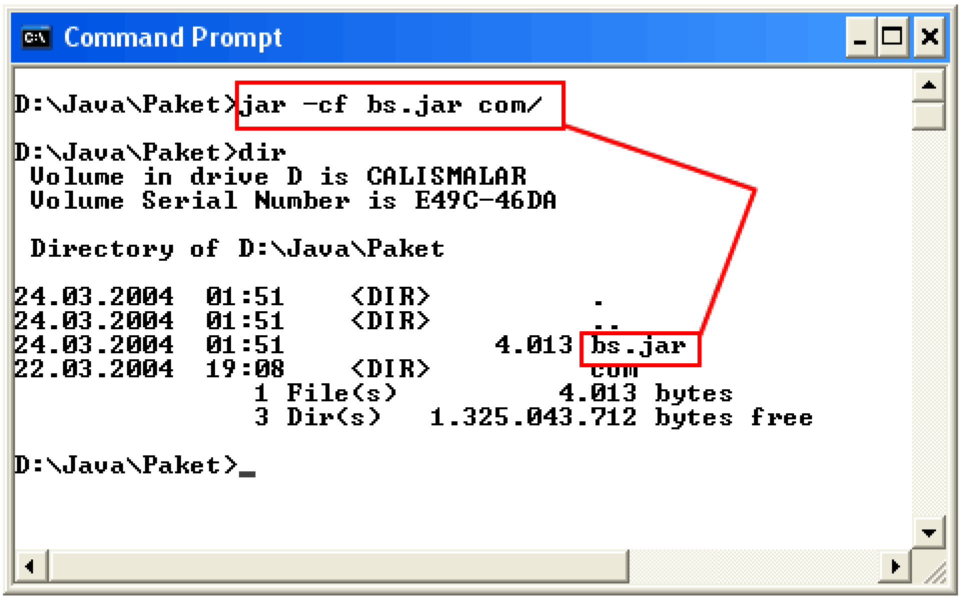
JAR Java Arşivi olarak tanımlayabileceğim bir dosya formatı. Bu dosyanın özelliği, içerisinde çeşitli paketleri barındırabilmesi. Hatta, paketler dışında, kaynaklardan edindiğim bilgiye göre, şu

an için sadece ne olduklarını bildiğim ama nasıl yapıldıklarını bilmediğim applet�ler için gerekli resim, ses dosyası gibi kaynaklarıda bünyesinde barındırabiliyor. Paketleri bu şekilde bir yerde toplamanın amacının onları bir arada düzenli bir şekilde tutmak olduğunu söyleyebiliriz. Ancak bununla birlikte, sıkıştırma kabiliyeti sayesinde, özellikle internet üzerinden gerçekleştirilen download işlemlerindede faydalı olduğunu söyleyebiliriz. Diğer yandan JAR dosyalarının bence en önemli özelliği, fiziki adres bağımlılığından kurtulmuş olunması. Bunun anlamı şu. Şu ana kadar yazmış olduğumuz paketleri belli klasörler içerisinde belirli bir sistematik içerisinde oluşturduk. Ancak paketleri JAR dosyasına aldıktan sonra, bu dosyayı sistemde her hangibir klasöre alabilir ve tek bir dosya içinden, tüm paketlerimize kolayca ulaşabiliriz. Ancak her zamanki gibi CLASSPATH ayarında bu kez JAR dosyasının konumunu belirtmeliyiz. Ne yazıkki bunuda söylemek durumunda kaldım. Açıkçası şu CLASSPATH�ten kurtulamamış olmak üzücü.

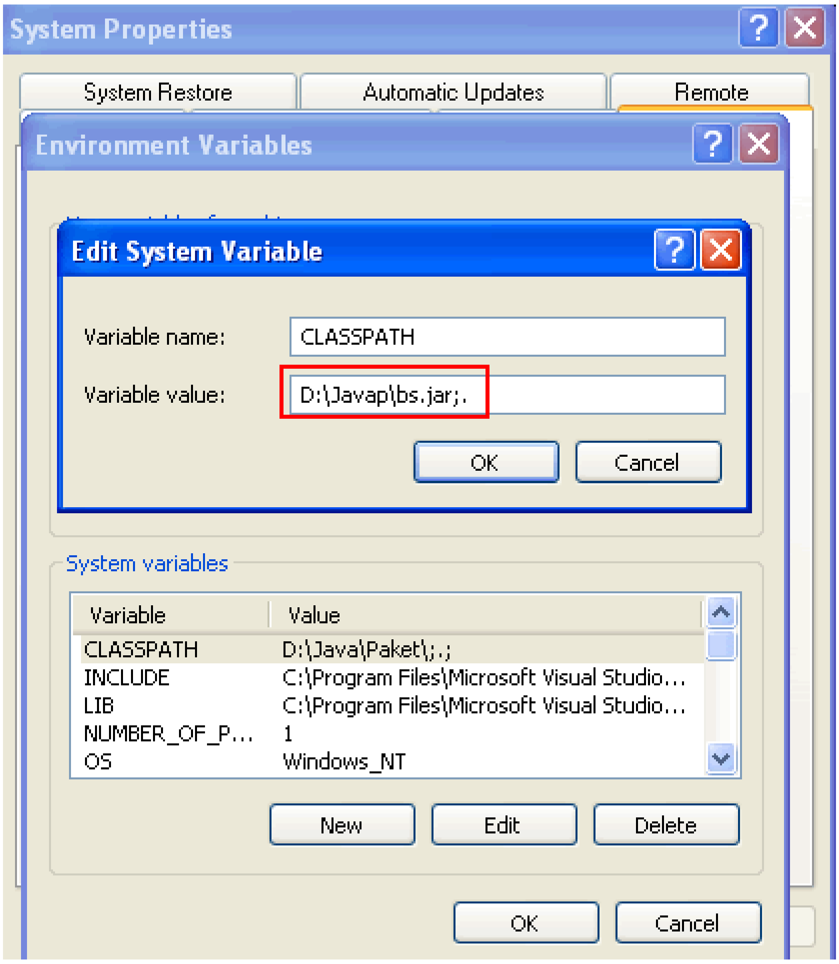
Bu bilgiler ışığında hemen oluşturduğum iki paketi tek bir JAR dosyası içerisine almak için girişimlerime başladım. Bunun için paket tanımlarının yer aldığı en üst klasöre çıkmam gerekiyor. Yani aşağıdaki şekilde görüldüğü gibi JAR dosyasını oluşturmak için gerekli komutumu Paket isimli klasör içinden vereceğim. Böylece, bu klasör altındaki tüm paketleri JAR dosyası içine almış olacağım.



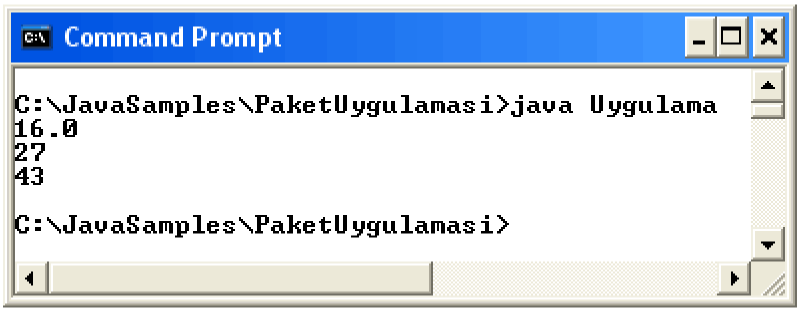
İşte JAR dosyası için kullandığım komut.



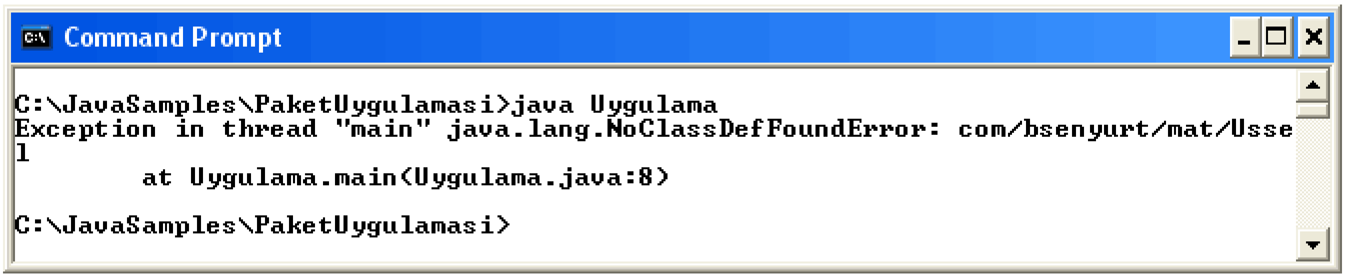
Görüldüğü gibi JAR dosyası başarılı bir şekilde oluşturuldu. Burada jar komut dizimindeki -cf parametresi, izleyen parametredeki isimde(bs.jar) bir JAR dosyası yaratacağımızı ve bu dosya içine com dizini altındaki tüm paketleri dahil edeceğimizi belirtiyor. Şimdi JAR dosyasının ününe ün katan işlevi gerçekleştirmek lazım. Bu dosyayı buradan alıyorum ve C: sürücüsünün hemen altındaki javap klasörü altına taşıyorum. Hemen ardından son yazmış olduğum uygulamayı çalıştırıyorum. Uygulama çalışıyor çalışmasınada şu noktada paketlere nereden bakıldığından emin değilim. Bir anda aklıma CLASSPATH tanımı geliyor. Hemen CLASSPATH tanımımı yeniden ayarlıyorum. Ancak bu kez bir klasörden ziyade JAR dosyasını bulunduğu fiziki adres ile birlikte belirtiyorum.



Lütfen CLASSPATH tanımının nasıl yapıldığına dikkat edelim. Önceki tanımlamalardan farklı olarak jar dosyasının açıkça belirtiyoruz. Şimdi uygulamayı tekrar çalıştırıyorum ve başarılı bir şekilde çalıştığını görüyorum.

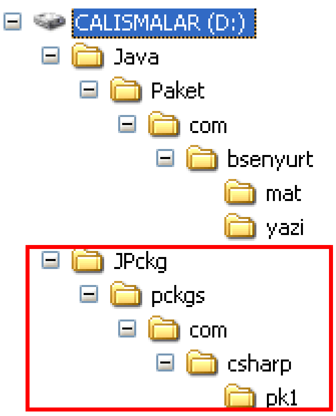


Şimdi aklıma kurnazca bir fikir geliyor. Bu paketleri JAR dosyası içine almak ile acaba, bu paketleri sistemden kaldırdığım zamanda uygulamam çalışıcakmı? Bu amaçla yola çıkarak birazda sinsi bir gülümseme ile, paketleri tanımlamış olduğum klasörleri sistemdeki başka bir yere taşıdım. Programı tekrar çalıştırdığımda ise, aşağıdaki hata mesajı ile karşılaştım.



Beklediğim gibi olmadı. Bir hata almamayı düşünüyordum. Sanırım beklentim, paketleri JAR dosyası içerisine almak ile, onları tüm içerikleri ile birlikte buraya alabileceğim yönündeydi. Ama bölye olmadı. Demekki JAR dosyalarını oluşturmakla paketleri bir yerde topluyorduk ancak paketlerin adreslerini kesinlikle değiştirmememiz veya silmememiz gerekiyordu. Bununla birlikte bu JAR dosyasının fiziki olarak sistemden istediğimiz yere taşıyabiliriz. Ama yinede CLASSPATH ayarlarını buna göre uyarlamamız gerekiyordu.

Diğer bir sorun ise şuydu. Sonradan yazılmış başka bir paketi bu JAR dosyasına nasıl alabilirdik. Pekala JAR dosyasını tekrardan üretebilirdik. Peki ya **paketimiz başka bir klasörde yer alıyorsa, bu paketi JAR dosyasına eklememiz ile, tek bir CLASSPATH tanımı sayesinde, sistemdeki farklı klasörlerde yer alan paketlere erişebilirmiydik acaba?** Güzel soru değil mi? Bunu öğrenmenin tek bir yolu var. Java\Paket\ altındaki paket tanımlarından farklı bir yerde bir paket oluşturmak, bunu bs.jar dosyasına eklemek ve uygulama içerisinde bu yeni pakete erişmeye çalışmak.

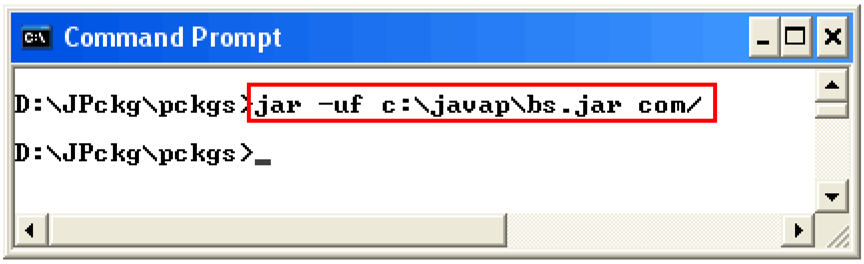


Yeni klasör içerisinde hemen com.csharp.pk1 paketini oluşturdum.

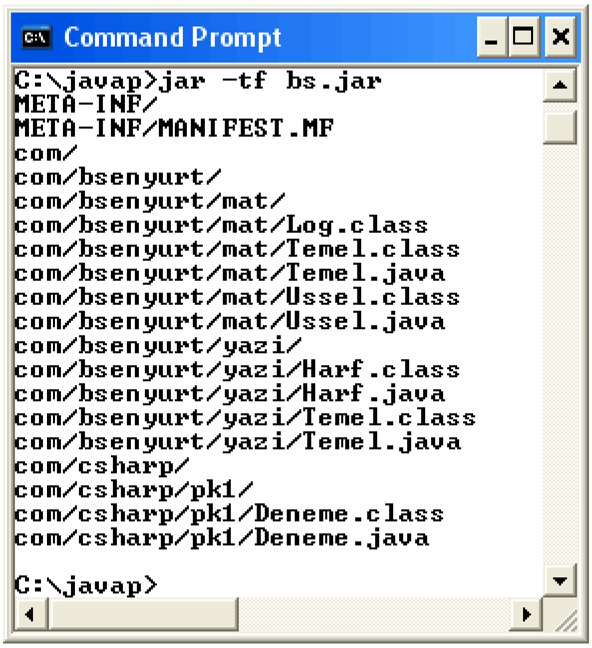
```java
package com.csharp.pk1;

public class Deneme
{
   public void Yaz()
   {
      System.out.println("com.csharp.pk1 paketi burada...");
   }
}
```

Java dosyasını başarı ile derledikten sonra, sıra bu paketi bs.jar dosyasına eklemeye geldi. Bunun için aşağıdaki komut dizimini kullandım.



Bu komut ile var olan JAR dosyamıza yeni bir paket daha ekleyebiliriz. Aslında -uf belirtilen dosyayı günceller. Şimdi jar dosyasının içeriğine baktığımızda, com.csharp.pk1 paketinin konumununda eklendiğini görürüz.



Şimdi bu yeni paketimizde yazdığımız sınıfımıza ait metodumuzu uygulamamızda kullanalım.

```
import com.bsenyurt.mat.*;
import com.bsenyurt.yazi.*;
import com.csharp.pk1.*;

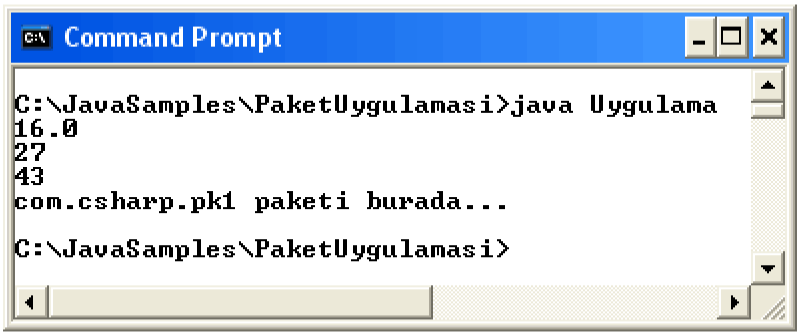
public class Uygulama
{
    public static void main(String[] args)
    {
        Ussel u1=new Ussel();
        double sonuc=u1.usal(2,4);
        System.out.println(sonuc);

        com.bsenyurt.mat.Temel t1=new
com.bsenyurt.mat.Temel();
        int toplam=t1.Toplama(12,15);
        System.out.println(toplam);

        com.bsenyurt.yazi.Temel t2=new
com.bsenyurt.yazi.Temel();
        t2.Uzunluk("Bu için tükettigim kaçinci kahve...");

        Deneme d1=new Deneme();
        d1.Yaz();
}
}
```

Uygulamayı derlediğimde, JAR dosyasına eklemiş olduğum yeni pakete ait sınıfında başarılı bir şekilde örneklendirildiğini ve Yaz metodunun çalıştırıldığını gördüm.

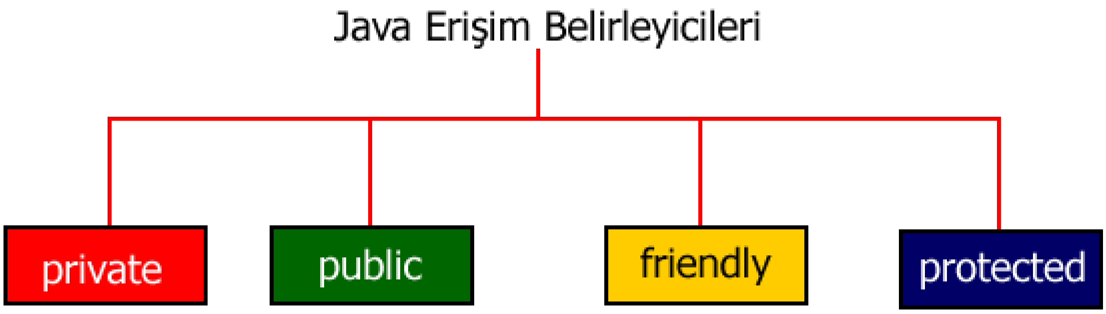


Hay allah yine kahvem bitti. Artık paket kavramınıda iyice işledikten sonra dinlenmenin zamanı geldi. Bir sonraki kahve molasında ise erişim belirteçlerini incelersem Java işinde biraz daha yol katetmiş olacağımı sanıyorum. En azından J harfinin şapkasını çizmiş olabileceğim.

Bölüm 6: Erişimleri Belirlemek (Access Specifiers)

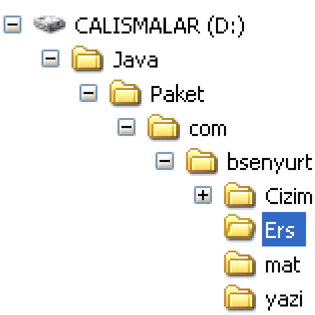
Sanıyorumki tüm nesne yönelimli programlama dillerinde en önemli kavramlardan birisi, oluşturulan nesnelerin birbirleri ile ilişkilerinin ne ölçüde olacağına karar verilmesidir. Bu kararı vermede, erişim belirleyicilerinin gerçekten çok önemli bir yeri var. Özellikle C# dili ile program yazarken, public, private gibi erişim belirleyicilerinin kullanımına özen gösteririm. Nitekim bunları doğru yerlerde doğru amaçlar için ve doğru yönlendirmeler ile kullanmak isterim. Diğer yandan, karmaşık ve kod satırlarının sayısı 10 binler ile ifade edilebilecek projelerde, kodu kitap okur gibi okuyabilmek, dilin temelinde yatan ancak çoğu zaman dikkat etmeden geçtiğimiz unsurların iyi bilinmesini gerektirir. İşte benim inancıma göre bu temellerden biriside erişim belirleyicileridir. Aslında sözü bu kadar uzatmanın anlamı yok. Hemen harekete geçmeli ve icraatlar yapmalıyım diye düşünüyorum.

Her nesne yönelimli programlama dili uygulayıcısının mutlaka ve mutlaka erişim belirleyicilerine aşinalığı vardır. Hatta %100 olduğundan eminim. Benim aşinalığım ise C# programlama dilinden geliyor. Bu hafta boyunca Java dilini öğrenmek için kendime ayırdığım vakitlerde, erişim belirleyicileri ile ilgili olarak kaynakları inceledim. Gerçekten Java dilinde'de çok büyük öneme sahip olduklarını ve iyi bilinmeleri gerektiğini keşfetmiş olduğumu büyük bir gururla söylemek isterim :) Java dilinde erişim belirleyicileri 4'e ayrılmaktadır. Bunları daha iyi anlayabilmek için özel bir renklendirme stratejisi uyguladığım aşağıdaki tabloyu oluşturdum.



Bu şekilde Java dilindeki erişim belirleyicilerini değişik renkler ile göstermeyi tercih ettim. Neden bu tarz bir renklendirme kullandığımı ise birazdan hep birlikte anlayacağız. Erişim belirleyicilerini sınıflara, metodlar ve alanlara uygulayabilmekteyiz. Elbette erişim belirleyicisinin tipine göre bir takım farklılıklarında olacağı söyleniyor. Anlaşılan bu farklılıkları örnekleri yazdıkça daha iyi kavrayacağım. Artık işe koyulmanın tam sırası. Sıcak bir fincan kahvemi yanıma alıyorum ve erişim belirleyicilerini tek tek incemeleye başlıyorum.

Öncelikle private erişim belirleyicisinden bahsedelim. Private isim anlamı ilede özel bir kullanıma sahiptir. Bu erişim belirleyicisi sadece metodlara ve alanlara uygulanabilir. Onu özel yapan, sadece tanımlanmış olduğu sınıf içerisinde kullanıma izin vermesidir. Bu, başka bir paketten, varsayılan paketten hatta bu private üyeleri içeren sınıfın bulunduğu aynı paketteki diğer sınıflardan bile erişimin gerçekleştirilemiyeceğini anlatır. Anlatır anlatmasına ama, bunu örnekler ile incelemeden anlamak şu an için zor olabilir. Bu nedenle hemen bir örnek geliştirmeye karar verdim. Her zaman olduğu gibi kodları işleyebilmek ve erişim belirleyicilerinin paketler arasındaki etkileşimini daha iyi anlayabilmek için örnek bir paket hazırladım.



Bu kez com.bsenyurt.Ers isimli bir paketi kullanacağım. İlk örneğimde, private alanları ve metodları içeren bir sınıfım var.

```
package com.bsenyurt.Ers;

public class PrivateD
{
   private int Deger;
   private void Metod()
   {
      Deger=10;
   }
}
```

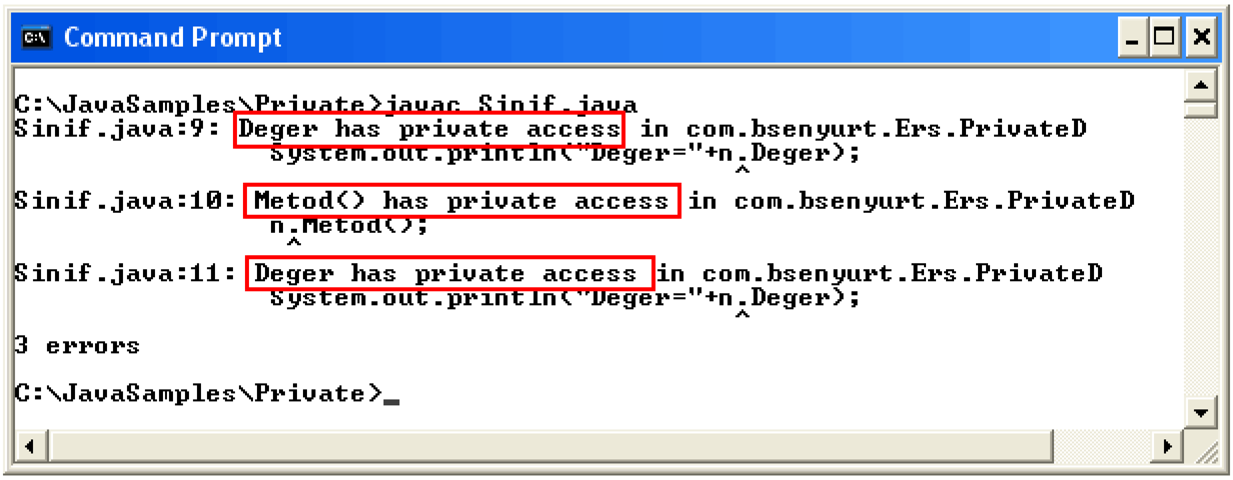
Bu paket içerisinde tanımladığım PrivateD sınıfı içinde Deger isimli bir private alan ve Metod isimli private bir yordam tanımladım. Private erişim belirleyicisinin uygulattığı sınırlamaya göre bu alan ve metoda sadece bu sınıf içerisinden erişebilirim. Nitekim, Deger alanının değerini Metod yordamı içinden değiştirebilmekteyim. Fakat private belirleyicisinin tanımı gereği öncelikle bu paket içindeki sınıfa ati private üyelere, bu paketi uygulayan başka bir sınıf içinden erişememem gerekiyor. Bunu ispat etmek zorunluluğunu elbette hissediyorum. İspat matematik bilmindede çok önemli bir yere sahip. Hatırlıyorumda üniversite yıllarında sayısız ispat yapardık. Sonuçta hipotezleri teoriye dönüştürür ve geçerliliği kanıtlanırdı. Aynı şey bence bir programlama dilini öğrenirkende geçerli. Verilen tanımlamaların ispatını yapmayı başarbilirsek o zaman dilde uzmanlaşmamız daha kolay ve güçlü olur. Sözü uzatmadan private alanların bu tanımı gereği ilk ispatı yapmak üzere bilgisayarımın herhangibir klasöründe bu paketi kullanacak bir sınıf daha yazdım.

```
import com.bsenyurt.Ers.*;

public class Sinif
{
   void Yaz()
   {
      PrivateD n=new PrivateD();
```

```
        System.out.println("PrivateD nesnesi oluşturuldu");
        System.out.println("Deger="+n.Deger);
        n.Metod();
        System.out.println("Deger="+n.Deger);
    }
}
```

Bu sınıf içinde, com.bsenyurt.Ers paketi içindeki PrivateD sınıfınından bir nesne örneği oluşturmaya ve, PrivateD sınıfı içindeki Deger alanı ile Metod yordamlarını çağırmaya çalıştım. Sonuç olarak, sınıfı derlemeye çalıştığımda aşağıdaki hataları aldım.



İspatın sadece ilk kısmı bitti ama. Sırada com.bsenyurt.Ers paketi içinde yer alan başka bir sınıf içinden, PrivateD sınıfı içindeki private üyelere erişebilip erişemiyeceğimi kontrol etmem gerekiyor. Tanım gereği bu şekildede private üyelere erişememem gerekli. Bu amaçla, com.bsenyurt.Ers paketi içinde başka bir sınıf tanımlıyorum.

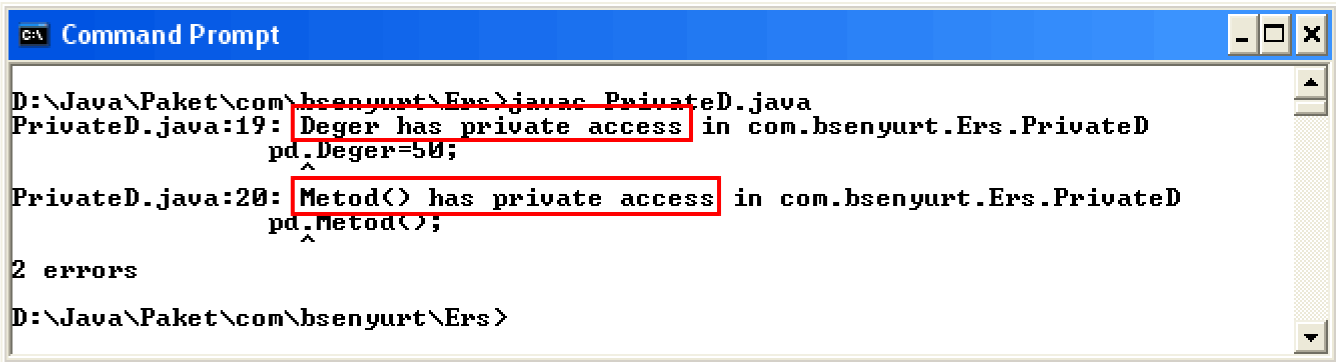
```
package com.bsenyurt.Ers;

public class PrivateD
{
    private int Deger;
    private void Metod()
    {
        Deger=10;
    }
```

```
}

class PSinif
{
    void Deneme()
    {
        PrivateD pd=new PrivateD();
        pd.Deger=50;
        pd.Metod();
    }
}
```

Paketimin içinde PSinif isimli yeni bir sınıf tanımladım ve bu sınıf içindeki Deneme metodundan PrivateD sınıfındaki özel üyelere erişmeye çalıştım. Sonuç beklediğim ve umduğum gibi oldu.



Anlaşılan şu ki private üyelere, başka bir paketten yada aynı paket içinden erişemiyoruz. Bu da private üyelerin sadece tanımlandıkları sınıf içinden erişilebildikleri anlamına geliyor. Privatizim adını vereceğim bir dogma burada kendini gösteriyor. Privatizim, aslında nesne yönelimli programlama dillerinin, kapsülleme kavramının temelinde yatan uygulanabilirliğin bir göstergesi. C# dilinde sınıf içindeki alanları dış ortamlardan soyutlamak istediğimizde bunları private tanımlar ve bu alanlara erişmek için özellikleri kullanırız. Aynı şekilde sadece sınıf içindeki özel alanlar ile ilgili işlemleri yapacak özel sınıflarda tanımlar ve bunlarıda dış ortamdan soyutlayarak kapsüllemeyi gerçekleştiririz. Aynı düşüncelere java dili içinde geçerli.

Private kullanımına ilişkin kaynak araştırmalarımı incelerken güzel bir örnek yakaladım. Bu örnekte, bir sınıfa ait yapıcı

metod private olarak tanımlanmıştı. Bunun doğal sonuçlarını oturup bir kaç saniye düşündüğümde, bu yapıcının tanımlandığı sınıfa ait bir nesne örneğinin başka paketler içindeki sınıflar içinden yada aynı paketteki başka sınıflar içinden tanımlanamıyacağını farkettim. O halde bu tarz bir nesne örneğini nasıl oluşturabilir ve alanlarına erişebilirdik. Herşeyden öte neden böyle bir gereklilik olsun ki. Sebebi gayet basit. Bazen yapmış olduğumuz bir sınıfın kesinlikle başka sınıflar içinden örneklenememesini isteyebiliriz. Sınıfımız gerçekten başlı başına özeldir ve bu nedenle nesne örneğinin oluşturulabilmesi yasaklanmıştır. Ancak bu sınıf içine koyacağımız static bir metod bizim belirleyeceğimiz kurallar çerçevesinde, bu nesne örneğinin oluşturulmasına ve kullanılmasına imkan sağlıyabilir.

Ne demek istediğimi son paragrafı tekrar okuduğumda bende tam olarak kestiremedim. İşte keskinliği arttırmanın yolu. Olayı gerçekçi bir örnek üzerinde düşünmek. Bu amaçla com.bsenyurt.Ers paketindeki PrivateD sınıfının tanımını biraz değiştirdim. Öncelikle private yapıcı metodu tanımladım.

```
package com.bsenyurt.Ers;

public class PrivateD
{
   private int Deger;
   private PrivateD()
   {

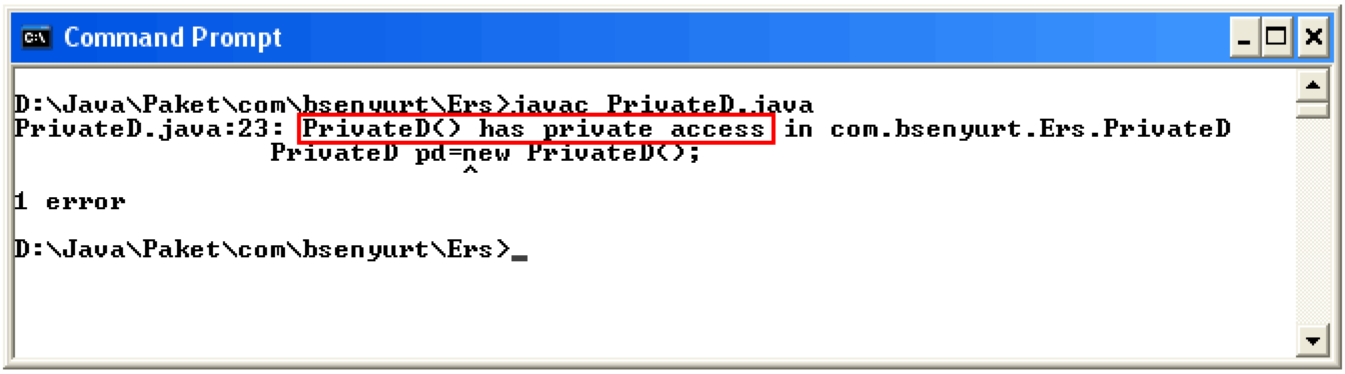
   }

   private void Metod()
   {
      Deger=10;
   }
}

class PSinif
{
   void Deneme()
   {
```

```
        PrivateD pd=new PrivateD();
    }
}
```

Öncelikle görmek istediğim, varsayılan yapıcının private tanımlanmasının, nesne örneğini oluşturmaya olan etkisi idi. Etki aşağıdaki gibi ekrana yansıyıverdi.



Peki o halde böyle bir sınıfı nasıl örnekleyebilirdim. Cevap biraz düşününce ortaya çıkıverdi. Static ve dışarıdan erişilebilir bir metod içinde bu nesneyi tanımlayabilir ve sınıf ile ilgili gerekli, istediğim düzenlemeleri bizzat sınıfı yazan kişi olarak ben yaptırtabilirdim. Böylece bu sınıfımın başkaları tarafından ancak benim belirlediğim kurallar çerçevesinde oluşturulabilmesini sağlardım. Bu düşüncemin sonucunda aşağıdaki örnek ortaya çıkıverdi.

```
package com.bsenyurt.Ers;

public class PrivateD
{
    private int Deger;
    private PrivateD()
    {

    }

    private void Yaz()
    {
        System.out.println(Deger);
    }
```

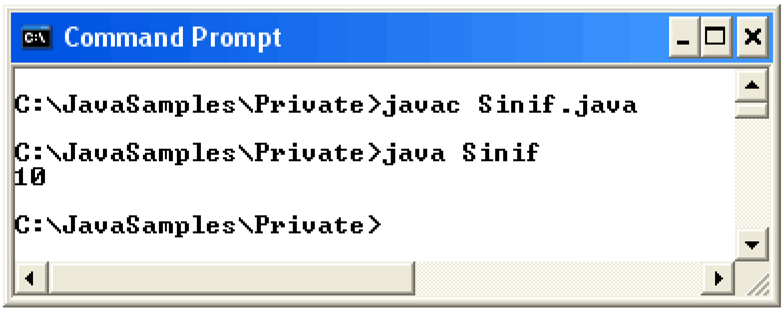
```
    public static void Kullan()
    {
        PrivateD pd=new PrivateD();
        pd.Deger=10;
        pd.Yaz();
    }
}
```

Şimdi bu paketi başka bir sınıfta kullanmayı denemem gerekiyor. Bu amaçlada aşağıdaki örneği geliştirdim.

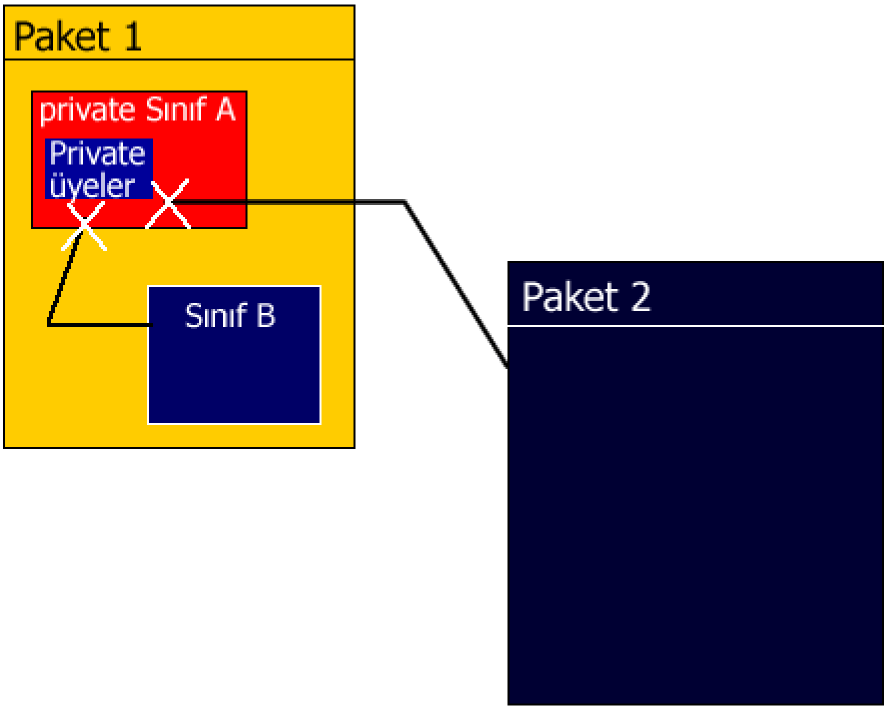
```
import com.bsenyurt.Ers.*;

public class Sinif
{
     public static void main(String[] args)
     {
        PrivateD.Kullan();
     }
}
```

Tabi burada göze çarpan benim public erişim belirleyicisini daha anlatmadan kullanmış olmam. Aslında public erişim belirleyicisini diğerlerini olduğu gibi C# dilinden biliyorum. Bu nedenle static olarak tanımladığım bu metoda dışarıdaki bir sınıftan erişebilmem için yani paket dışından halka açık şekilde tanımlamam gerekliydi. Ama yazının ilerleyen kısımlarında bu erişim belirleyicisinide detaylı bir şekilde inceleyeceğim. Ne de olsa gözden bir şey kaçırmamak lazım. Uygulamayı başarılı bir şekilde derledikten sonra çalıştırdım ve aşağıdaki ekran görüntüsünü elde ettim.



Private ile ilgili olarak söylenecek tek bir şey kaldı. Aslında söylemekten ziyade şekil olarak private erişim belirleyicisinin nasıl işlediğini daha iyi görselleyebilmek. Bu amaçla önce kara kalemimi aldım ve kağıt üzerinde bir çizim yaptım. Kendimle mutabakata vardıktan sonra ise, bu örneği dijital ortama geçirdim. Sonuçta aşağıdaki grafik ortaya çıktı. Ha bu arada ilk şekilde private erişim belirleyicisini neden kırmızı renk ile gösterdiğimi sanırımı anlamışsınızdır. Bu renk genelde erişimi kısıtlanmış olan durumlar için kullanılır. Ancak kastettiğim elbetteki kırmızı trafik lambasıda değil.



Şimdi sıra geldi friendly erişim belirleyicisini incelemeye. Private erişim belirleyicisinden ziyade, friendly erişim belirleyicisi Java dilinde varsayılan erişim belirleyicisidir. Yani bir üye için erişim belirleyicisi belirtilmesse bu üye dost canlısı olarak kabul edilir. Bununla birlikte friendly erişim belirleyicisi, alanlara, metodlara ve sınıflara uygulanabilir. Özelliği ise şudur. Bu tipteki üyelere aynı paket içinde erişilebilirken farklı paketlerden erişilememektedir. Elbette bu tanımı ispat etmem gerekiyor. Haydi bakalım kolları sıvamanın zamanı geldi. Bir yudum kahve dopingi ve ardından işte aşağıdaki örnek.

```
package com.bsenyurt.Ers;
```

```
class KardesSinif
{
   int Deger;
   void Degistir(int yeniDeger)
   {
      Deger=yeniDeger;
   }
}
```

Örnek sınıfı yine com.bsenyurt.Ers paketi içinde oluşturdum. Burada sınıf için, int tipteki Deger değişkeni için ve Degistir isimli metod için herhangibir erişim belirleyicisini kullanmadım. Böylece bu üyelerin friendly erişim belirleyicisinine sahip olmasını sağlamış oluyoruz. Tabi aklıma saf ve yeni bir java programcısı olarak, bu üyelerin başına friendly anahtar kelimesini eklemekte gelmedi değil. Bunu yaptığım zaman yani örneği aşağıdaki şekilde değiştirdiğimde, java derleyicisi bana **" Ben herşeyin farkındayım, zaten bir şey yazmasan friendly olduklarını anlarım, beni boşu boşuna bir de bu yazdıklarını denetlemek zorunda bırakmanı anlamıyorum"** gibisinden bir hata mesajı verdi.

```
friendly package com.bsenyurt.Ers;
class KardesSinif
{
   friendly int Deger;
   friendly void Degistir(int yeniDeger)
   {
      Deger=yeniDeger;
   }
}
```

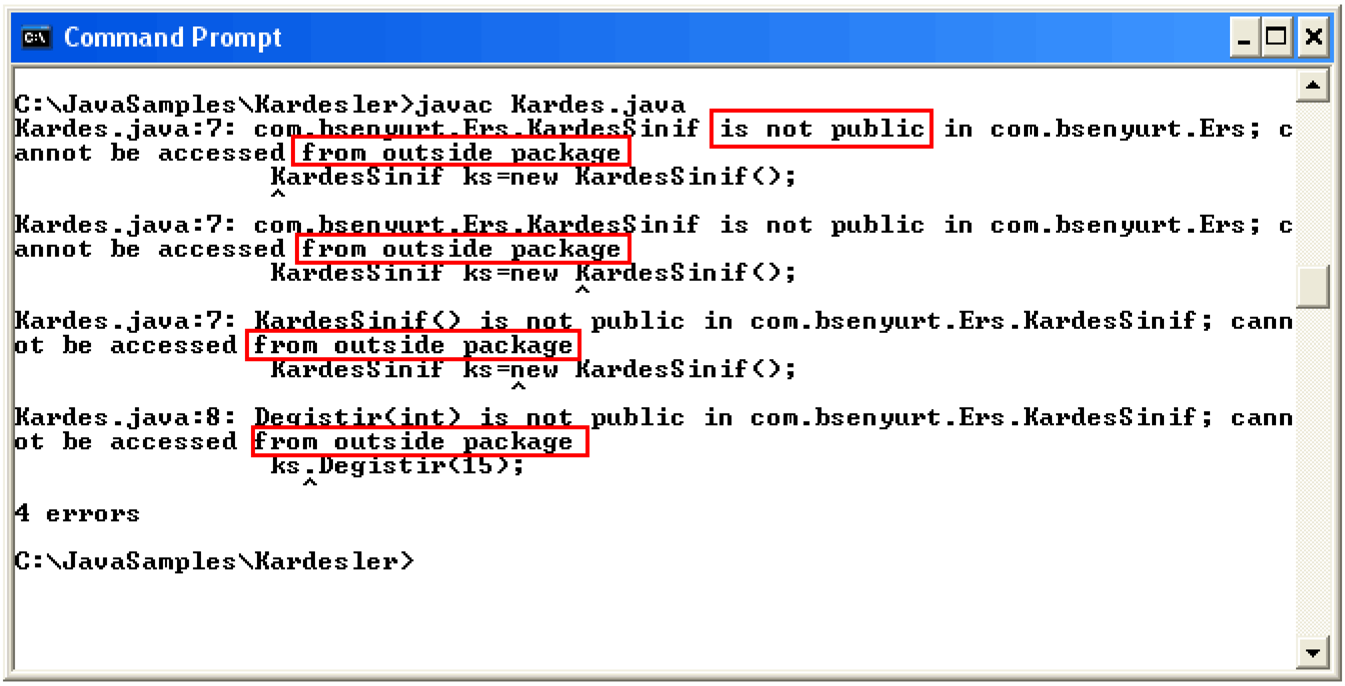
Ve java derleyicisinin karşılık olarak verdiği tarihi cevap .

```
D:\Java\Paket\com\bsenyurt\Ers>javac KardesSinif.java
KardesSinif.java:1: 'class' or 'interface' expected
friendly package com.bsenyurt.Ers;
^
KardesSinif.java:6: <identifier> expected
        friendly int Deger;
                ^
KardesSinif.java:8: <identifier> expected
        friendly void Degistir(int yeniDeger)
                ^
KardesSinif.java:11: ';' expected
        }
         ^
KardesSinif.java:6: cannot resolve symbol
symbol  : class friendly
location: class KardesSinif
        friendly int Deger;
        ^
KardesSinif.java:8: cannot resolve symbol
symbol  : class friendly
location: class KardesSinif
        friendly void Degistir(int yeniDeger)
        ^
6 errors
```

Boyumun ölçüsünü aldıktan sonra örneği eski haline getirdim. Daha sonra sistemin her hangibir klasörü içinde com.bsenyurt.Ers paketindeki KardesSinif sınıfını kullanacak bir örnek oluşturdum. Teori eğer doğru ise, bu yeni örnek sınıfı içinden, KardesSinifi üyelerine erişememem gerekiyor. Çünkü yeni örnek sistemin herhangibir yerinde varsayılan paket özellikleri ile oluşturulacak. Bu amaçla aşağıdaki örneği geliştirdim.

```
import com.bsenyurt.Ers.*;
class Kardes
{
    public static void main(String[] args)
    {
        KardesSinif ks=new KardesSinif();
        ks.Degistir(15);
    }
}
```

Örnek elbette derlenmedi. Gerçektende, com.bsenyurt.Ers paketi içindeki KardesSinifi sınıfını friendly olması nedeni ile kullanamamıştım.



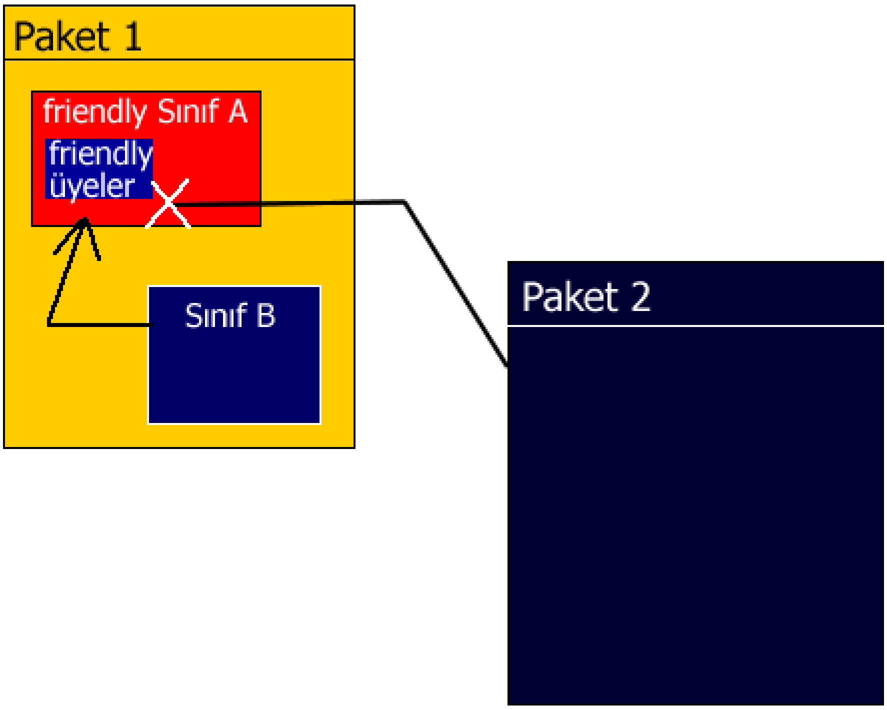
Anlaşılan friendly erişim belirleyicisine sahip üyeler aile dışına çıkmayı pek sevmiyorlar. Aileden kastım elbette bu üyelerin bulunduğu paket. Dolayısıyla Teorinin ikinci kısmına bakmak gerekiyor. Buna göre, aynı paket içinden friendly üyelere erişilebiliyor. Bu amaçla, com.bsenyurt.Ers paketi içinde aşağıdaki sınıfı oluşturdum. Bu sınıf içinden açıkça, KardesSinif sınıfına erişmeye çalışılıyor. KardesSinif sınıfından ks isminde bir nesne örneği yaratılıyor ve bu nesne üzerinden Degistir isimli metod çağırılıyor.

```
package com.bsenyurt.Ers;

public class Kardes
{
    public void Kullan()
    {
        KardesSinif ks=new KardesSinif();
        ks.Degistir(15);
    }
}
```

Bu kez derleme işlemi başarılı bir şekilde gerçekleştirildi. Burada herhalde el alışkanlığından olsa gerek, Kardes isimli sınıf tanımının başına ve Kullan isimli metodun başına public erişim belirleyicisini ekledim. Evet aslında public ile ilgili teorileri C#

’tan bilmeme rağmen, Java’da nasıl olduğunu merak ediyorum. Ancak herşeyden önce, friendly ile ilgili son bir şey yapmam gerekiyor. Friendly erişim belirleyicisinin tanımını şekillendirmek.



Private ve friendly erişim belirleyicilerinin kısıtlamalarından sonra şöyle kendimi özgür hissettirecek bir erişim belirleyicisine ihtiyacım olduğunu düşünmeye başladım. Bu arzumun karşılığı sanıyorumki public erişim belirleyicisi. Public erişim belirleyicisine sahip sınıflar, metodlar ve alanlara, herhangibir paketten, aynı paketten, varsayılan paketten kısaca heryerden erişilebilir. Bu kulağa hoş gelen bir tanım olmakla birlikte, bazen programcıların ağır bedeller ödemesinede neden olmuştur. Bir üyeyi herkese açmak güzel bir fikir gibi görünebilir. Ancak burada durum, Linux gibi açık kaynaklı bir işletim sisteminin gelişiminden çok daha farklıdır.

Çoğu zaman iş uygulamalarında sınıfların belli başlı üyelerini hatta sınıfların kendilerini dış dünyaya kapatmak ve denetimi, kontrolü tamamen ele almak isteriz. Bazen sadece belli başlı üyelerin herkese açık olmasını isteriz. Örneğin bir bankacılık uygulaması için, banka müşterilerinin birer sınıf örneği olarak temsil edildiklerini düşündüğümüzde, sadece yetkili personelin bu kişiye ait bilgileri görmesi gerekecektir. Hatta müşterinin kendisi bile sınıf tanımı içindeki, hesap no yada bakiye gibi

bilgilere ulaşmak için, şifre ve daha fazlasına ödemek zorundadır. İşte bu gibi bir iş uygulaması üyelerin son derece iyi planlanmasını gerektirir. Bir müşteri sınıfını herkese kapatabilirsin. Ancak, bu sınıf içindelki özel bilgileri private yapmak, bu sınıfın yer aldığı paketler üzerinde çalışan programcıların kullanabilmesi için friendly üyeler açmak ve şifre sorgulama yada kullanıcı doğrulama gibi işlemleri herkesin kullanımına açan public metodlar uygulatmak daha mantıklıdır.

O nedenle public hem iyidir hemde iyi değildir. Bu nedenle her zaman inandığım bir şey vardır. Bir uygulama nesne yönelimli programlama dilleri üzerinde koşacaksa, onu koşturmadan önce gerçekten iyi tasarlanmalıdır. Yapılan herşey, belkide 3 kez kendimize sorulup, 3 kez program ekibine sorulup ve üç kezde genel müdüre sorulup ondan sonra yürürülüğe girmelidir. Evet 3 kez sormak biraz abartı gelebilir ancak 3 güvenlik görevlisinin, genel müdürün, program ekibinin ve hatta sayısız 3'ün bir araya gelerek oluşturduğu kalabalık bir müşteri grubunun bizi kovalamasından iyidir. Neyse bu kadar abartı senaryolar gözümüzü korkutmamalı değil mi? Sonuç olarak Java'yı bu kahve molasında ilerletmek zorunluluğunu hissediyorum. O halde beklemenin anlamı yok. Sırada public erişim belirleyicisi var.

Aslında public erişim belirleyicisi ile ilgili söylenecek fazla bir şey yok. Bu erişim belirleyicisini incelemek için com.bsenyurt.Ers paketi içine aşağıdaki sınıfı ekledim.

```
package com.bsenyurt.Ers;

public class Topla
{
   public int Toplam(int d1, int d2)
   {
      return d1+d2;
   }

   public int Eleman=10;
}
```

Şimdi ilk denemek istediğim bu sınıf içindeki public üyelere aynı paket içinden erişip erişemiyeceğim. Bu amaçla, com.bsenyurt.Ers paketi içinde yer alan bir sınıfı aşağıdaki şekilde düzenledim.

```
package com.bsenyurt.Ers;

public class PSinif
{
    public void ToplamAl()
    {
        Topla t=new Topla();
        int t1=t.Toplam(10,11);
        int t2=t.Toplam(12,12);
        int t3=t.eleman;
        t.eleman=1;
    }
}
```

PSinif sınıfı içinden, aynı pakette yer alan, Topla isimli sınıfın, public üyelerine erişmeye çalıştım. Bu sınıfı derlediğimde herhangibir hata mesajı almayışım, bu erişimin geçerli olduğunu göstermekteydi. Sırada, bu public üyelere başka bir paket içerisinden erişip erişemiyeceğimin ispatı var. Bunun için, com.bsenyurt.Yazi paketi içindeki Temel sınıfını kullandım. Bu sınıfın kodlarına, com.bsenyurt.Ers.PSinif sınıfındaki ToplamAl metodunun aynısını ekledim.

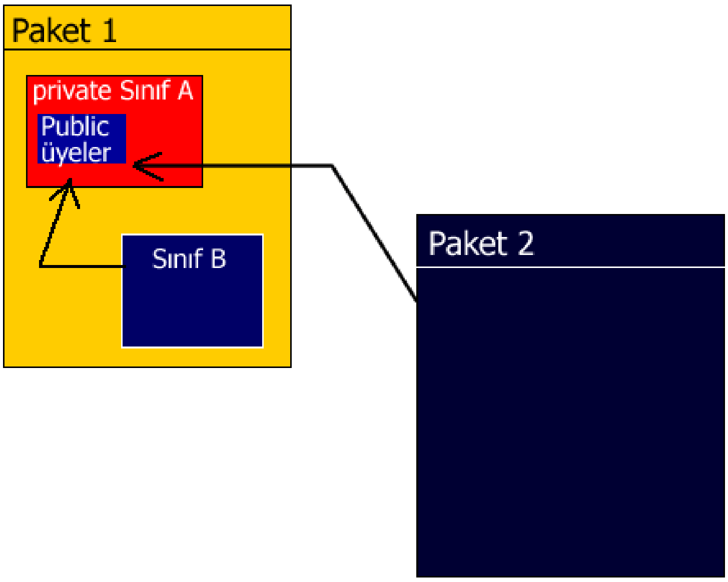
```
package com.bsenyurt.yazi;
import com.bsenyurt.Ers.Topla;

public class Temel
{
    public void Uzunluk(String metin)
    {
        System.out.println(metin.length());
    }

    public void ToplamAl()
    {
```

```
        Topla t=new Topla();
        int t1=t.Toplam(10,11);
        int t2=t.Toplam(12,12);
        int t3=t.eleman;
        t.eleman=1;
    }
}
```

Sonuçta, bu sınıfta başarılı bir şekilde derlendi. Yani, public üyelere, başka bir paket içindende erişebilmiştik. Aşağıdaki şekil sanıyorumki Public erişim belirleyicisinin davranışını çok daha net açıklıyor.



Erişim belirleyicilerinde son olarak protected erişim belirleyicisi kaldı. Bu erişim belirleyicisi aslında kalıtım kavramı ile ilişkili. Hazır kalıtım demişken bir sonraki kahve molasında bu konuyu işlemeye çalışacağım. İşte o zaman, protected erişim belirleyicisinin işlevini daha iyi anlayabileceğimi sanıyorum.

Kahvemden son bir yudum aldığımda aklıma, erişim belirleyicilerinin etki alanlarını gösteren bir tablo yapmak geldi. Bunun için bir süre uğraşmam gerekti ama sonunda başardım. Bu tablo erişim belirleyicilerinin etki alanlarını tam olarak açıklayabiliyor. Bu tablo aynı zamanda, buraya kadar işlediklerimin özetlenmesindede bana oldukça yardımcı oldu.

		Aynı Paket	Farklı Paket	Tür. Paket
Sınıf	public			
	friendly			
Metod	public			
	private			
	friendly			
	protected			
Alan	public			
	private			
	friendly			
	protected			

Bu tablo erişim belirleyicilerinin etki alanlarını kolayca anlamama yaradı. Kırmızılar erişimin olmadığını, yeşiller ise erişim olduğunu gösteriyor elbette. Tablonun okunması ise son derece kolay. Örneğin, Sınıfları ele alalım. Sınıflarda dikkat edilmesi gereken nokta, protected yada private sınıfların tanımlanamadığı. Diğer yandan public bir sınıfa her yerden erişebiliyoruz. Aynı paket içinden, farklı bir paket içinden, hatta türetilmiş bir paket içindende. Lakin friendly olarak yani dost canlısı bir sınıf bildirdiğimizde bu sınıfa sadece aynı paket içinden erişilebilmekte. Bu okuma şekli alanlar ve metodlar içinde geçerli.

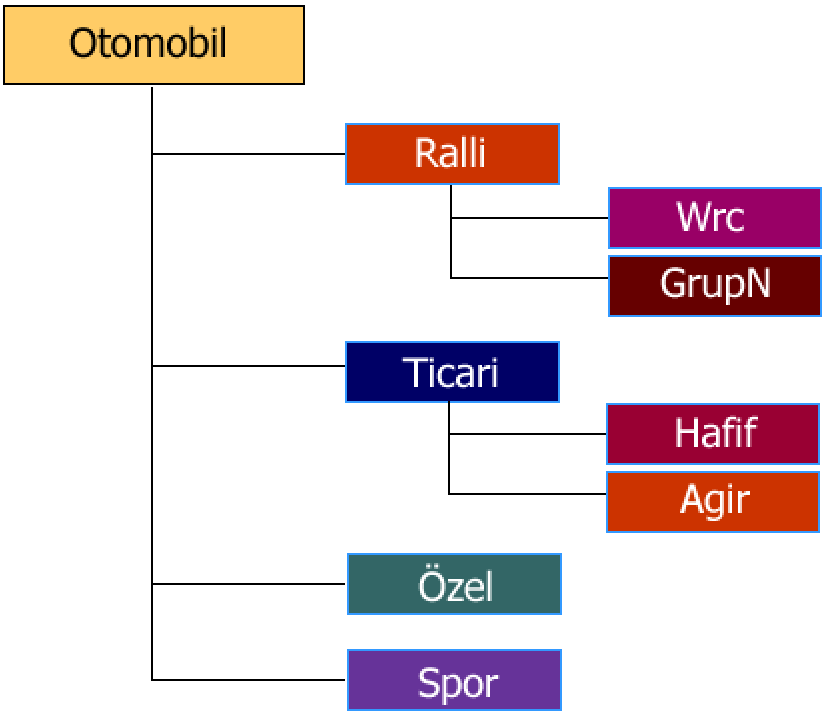
Bölüm 7: Kalıtım (Inheritance)

İşte nesne yönelimli programlama dillerinin en önemli kavramlarından birisi. Kalıtım. Normalde bu kavramı herkes gerçek hayattan biliyor. En basit anlamda, örneğin ben, annemin gözlerini almışım dediğimde, tıp uzmanlarının buna getirdikleri yorum " siz annenizden kalıtımsal olarak şu özelikleri almışsınız" oluyor. Programlama dillerinde de kalıtımın rolünün aynı olduğunu söyliyebilirim. Zaten nesne yönelimli

programlama dillerini tasarlayan uzmanlar, gerçek hayat problemlerini, bilgisayar ortamına taşıyabilmek amacıyla en etkili modelleri geliştirmişler. İşte bu model içerisine kalıtımıda katarak çok önemli bir özelliğin kullanılabilmesini sağlamışlar. Her şey iyi güzel de bu kalıtım kavramının programlama dilleri içerisinde bir tanımını yapmak lazım. En genel tanımı ile kalıtım, **"bir sınıftan yeni sınıflar türetmektir"** diyebilirim.

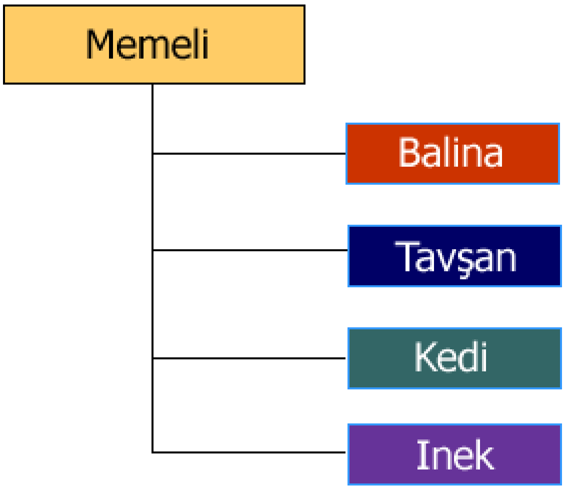
Bu genel kavramın arkasında elbette pek çok şey söylenebilir. Herşeyden önce kalıtım yolu ile bir sınıftan, yeni sınıflar türetilebilmesinin, türetilen sınıflara etkisi nedir? Bu sorunun cevabı kalıtımında özünü oluşturmaktadır. **Türetilen her bir sınıf, türediği sınıfın özelliklerinide devralır**. Buradan, türetilmiş bir sınıf içerisinden, türediği sınıfa ait üyelere erişilebileceği sonucunu çıkartabiliriz. Elbette bu erişiminde bazı kuralları vardır. Örneğin erişim belirleyicilerinin etkisi veya aynı üyelerin kullanılışı gibi durumlar.

Bu temel bilgiler ışığında öncelikle işe basit bir kalıtım senaryosu ile başlamam gerektiğini düşünüyorum. Neden bir sınıftan başka sınıflar türetiriz ki? Bunun cevabı son derece güzel. Tüm sınıflarda ortak olan özellikleri tek bir sınıf içerisinde toparlamak. Bu modellerimizi geliştirirken, her sınıf için ortak olan üyelerin tekrar yazılmasını engellemekle kalmıyacak, sınıflar arasında düzenli bir hiyerarşi yapısının oluşmasınıda sağlayacak. Şimdi güzel bir örnek lazım bana. Gerçek hayat modelleri bu iş için biçilmiş kaftan. Örneğin, otomobilleri bir temel sınıf olarak düşünebiliriz. Bu sınıftan otomobillere ait değişik kategorileri türetebiliriz.



İşte basit bir örnek. Buradaki tüm sınıfların ortak bir takım özellikleri var. Bir motorlarının olması, tekerleklerinin olması, viteslerinin olması vb. Ama aynı zamanda her ayrı sınıfın kendine has özellikleride var. Örneğin ralli araçları için güvenlik bariyerlerinin olması, pilotlar için kaskların kullanılması gibi. Bu tabloyu inceleyince, her ralli aracı bir otomobildir diyebiliriz. Bu ralli araçlarının otomobil sınıfından türediğini gösterir. Diğer yandan her wrc bir ralli aracıdır da diyebiliriz. Bu ise, wrc araçlarının ralli araçlarının bir takım ortak özelliklerine sahip olduğunu ayrıca otomobillerinde bir takım ortak özelliklerine sahip olduğunu gösterir. İlk aşamada, Ralli, Ticari, Özel ve Spor sınıflarının Otomobil sınıfından türediğini söyleyebiliriz. Bununla birlikte WRC ve GrupN sınıflarıda Otomobil sınıfından türeyen Ralli sınıfından türemiştir. Yani burada şunu söyleyebilmek mümkündür. WRC sınıfı hem Ralli sınıfının hemde Otomobil sınıfının özelliklerine kalıtımsal olarak sahiptir.

Otomobil sınıfı dışında başka örneklerde verebiliriz. Örneğin, değerli dostum Sefer Algan'ın Her Yönüyle C# isimli kitabında kalıtım için verdiği Memeli hayvanlar örneği gibi.



İki sınıf arasında kalıtım özelliğinin olduğunu anlayabilmek için **is-a** adı verilen bir ilişkinin kullanıldığını farkettim. Yani, **the cat is a mammiferous. Kedi bir memelidir.** Bu ilişkiyi yakaldıysak işte o zaman kalıtımdan söz edebiliyoruz. Yukarıdaki örnekte olduğu gibi.

Gelelim kalıtımın java sınıflarında nasıl uygulandığına. Bu amaçla hemen bir sınıf oluşturuyorum. Konu kalıtım olduğu için aklıma hemen sevgili Temel geliyor. Beni her zaman neşelendiren Karadenizli Temel. Kalıtımın nasıl uygulandığını görmek için Temel isimli ana sınıf bence biçilmiş kaftan. Değerli Temel aslında burada bizim için önemli bir tanımın temelini oluşturuyor aynı zamanda. Kalıtım senaryolarında, türetmenin yapıldığı en üst sınıflar Temel Sınıf(base class), bu sınıftan türetilen sınıflarda Tureyen Sınıf(derived class) olarak adlandırılıyor. Bu kısa bilginin ardından hemen kodlarımı yazmaya başladım.

```
class Temel
{
    public void Kimim()
    {
        System.out.println("Ben Temel'im");
    }
}
```

Hemen ardından bu sınıftan başka bir sınıf türetiyorum

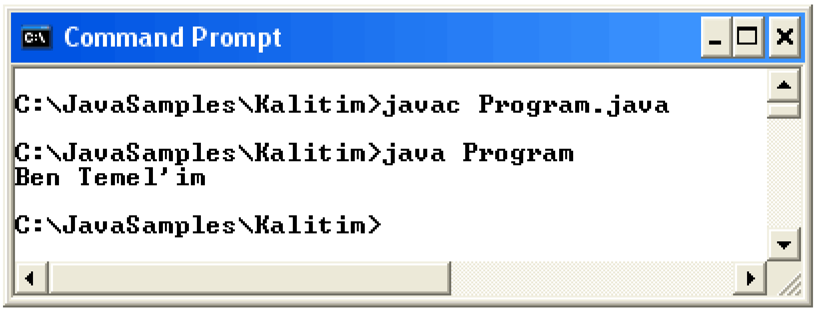
.

```
class Tureyen extends Temel
{
   public void Ben()
   {
      System.out.println("Ben Türeyen'im");
   }
}
```

Türetme işi java programlama dilinde extends anahtar sözcüğü ile yapılıyor. Aslında C# dilinde bu tanımlamayı yapmak daha kolay. İki nokta üst üste işareti ile :) Şimdide bu sınıfları uygulama içinden bir kullanmak lazım. İlk olarak merak ettiğim konu Tureyen sınıfa ait bir nesne üzerinden, temel sınıftaki bir üyeye erişip erişemiyeceğim.

```
public class Program
{
   public static void main(String[] args)
   {
      Tureyen turemis=new Tureyen();
      turemis.Kimim();
   }
}
```

Bu amaçla Tureyen sınıfa ait bir nesne örneği oluşturdum ve bu nesne örneği üzerinden Temel sınıf içinde yer alan Kimim isimli metoda erişmeye çalıştım. İşte sonuç.



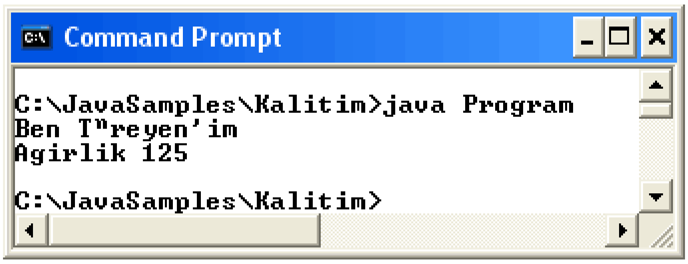
İşte kalıtımın doğal sonucu olarak, temel bir sınıftan türetilmiş bir nesne üzerinden, temel sınıftaki ortak metoda erişme imkanına sahip oldum. Bu noktada aklıma object sınıfı geliverdi. C# programlama dilinden biliyordumki, Object sınıfı en tepedeki sınıftı ve diğer tüm sınıfların atasıydı. Acaba java dilindede bu

böylemiydi? Bunu görmenin en kolay yolu object sınıfına ait toString metodunu herhangibir sınıfa uygulamaktı. Bu amaçla türemiş sınıf içerisinde toString metodunu kullanmayı denedim.

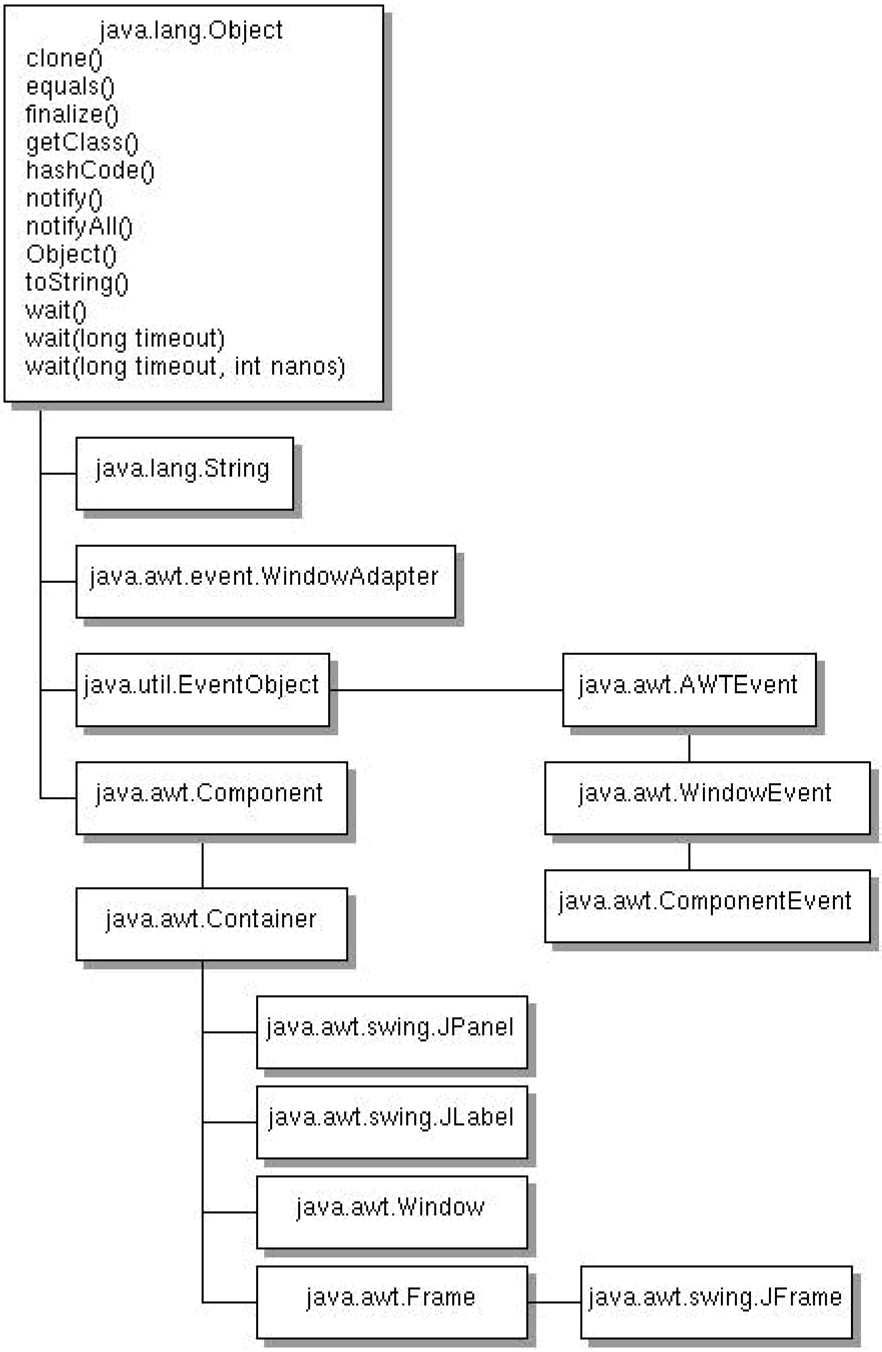
```
class Temel
{
   public void Kimim()
   {
      System.out.println("Ben Temel’im");
   }
}

class Tureyen extends Temel
{
   public void Ben()
   {
      Integer agirlik=new Integer(125);
      System.out.println("Ben Türeyen’im");
      System.out.println("Agirlik "+agirlik.toString());
   }
}
public class Program
{
   public static void main(String[] args)
   {
      Tureyen turemis=new Tureyen();
      turemis.Ben();
   }
}
```

Uygulamada, yeni bir integer değişken tanımlayıp bu değişken üzerinden object sınıfının toString metodunu çalıştırdım. toString metodu, sayısal değeri string türüne dönüştürmekteydi. Uygulamayı çalıştırdığımıda aşağıdaki sonucu elde ettim.



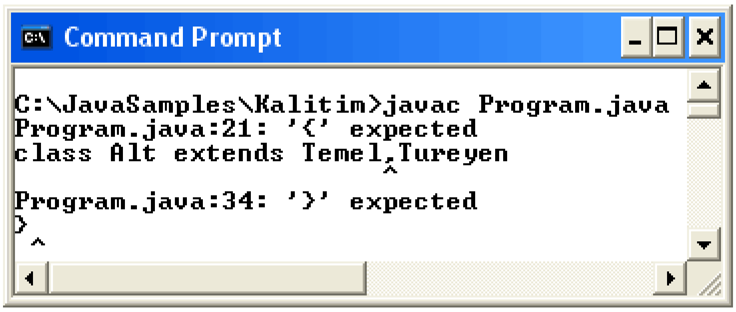
Programın çalışmasında özel veya değişik bir sonuç yoktu. Ancak önemli olan, bir sınıf nesnesinden object sınıfının metodlarına erişebilmiş olmamdı. Kaynaklardan edindiğim bilgiye göre buna gizli türetme ismi veriliyor. Bu, oluşturulan her sınıfın object sınıfından gizlice türetildiği ve bu nedenlede object sınıfına ait temel metodlara ulaşılabildiğini ifade etmekte. Elbette bu Object sınıfındaki üyelerin, bu sınıftan türeyen her sınıf için kullanılabilirliğini açıklıyordu. Bu konu ile ilgili olarak, güzel bir kaynakta aşağıdaki resmi elde ettim. Buna göre object sınıfı javadaki tüm sınıfların atasıydı. Saygı duyulması gereken bir sınıf olarak, hiyerarşinin en tepesinde yer almaktaydı.



Örneğin, ComponentEvent sınıfı, WindowEvent sınıfından, WindowEvent sınıfı, AWTEvent sınıfından, AWTEvent sınıfı EventObject sınıfından ve son olarakta EventObject sınıfıda Object sınıfından türetilmişlerdi. Bu noktada aklıma böyle bir hiyerarşide ComponentEvent sınıfının Object sınıfından itibaren nasıl türetildiği geldi. Acaba bir sınıf birden fazla sınıftan türetilebilirmiydi? Nitekim yukarıdaki şekli hiyerarşik yapısı ile göz önüne almadığım zaman böyle bir sonuç ortaya çıkıyordu. Bunu anlamın en güzel yolu, bir sınıfı bir kaç sınıftan türetmeye çalışmaktı. Bu amaçla aşağıdaki gibi bir bildirim denedim.

```
class Alt extends Temel,Tureyen
{

}
```



Aldığım hata mesajı tam olarak açıklayıcı değildi aslında. Ancak derleyici virgül yerine { küme parantezini bekliyordu. Bu aslında bir ipucuydu. Çünkü virgül yerine küme parantezinin istenmesi, bu noktadan itibaren direkt olarak sınıf bloğu istendiğini gösteriyordu. Dolayısıyla virgül notasyonu bir işe yaramamıştı. Ancak diğer taraftan, dayanamayıp kaynaklara baktığımda, java dilindede, C# dilinde olduğu gibi sınıflar arası çoklu kalıtımın desteklenmediğini öğrendim. Tahmin ettiğim gibi, çoklu kalıtımı uygulayabilmek amacıyla arayüzler(Interfaces) kullanılacaktı.

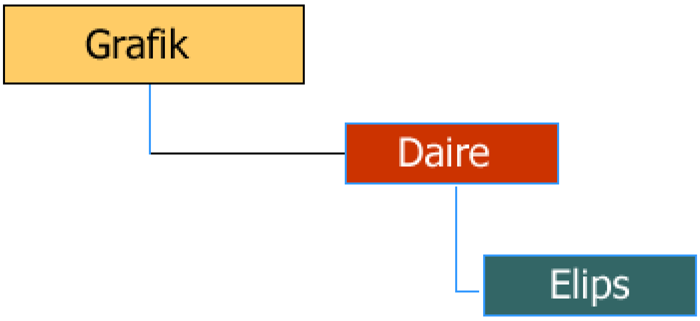
Kalıtım ile ilgili bir diğer önemli konu ise yapıcı metodların bu işteki paylarıydı. Bir kalıtım hiyerarşisi içerisinde acaba en alttaki sınıfa ait bir nesnenin oluşturulmasının, bu sınıfın türediği sınıfların yapıcıları üzerinde ne gibi bir etkisi olabilirdi? Bunu görmek için, aşağıdaki gibi bir uygulama geliştirdim. Bu kez alt alta üç sınıf türettim. Her bir sınıfın varsayılan yapıcılarını düzenledim.

```
class Grafik
{
   public Grafik()
   {
      System.out.println("Grafik SINIFI YAPICISI");
   }
}
class Daire extends Grafik
{
   public Daire()
```

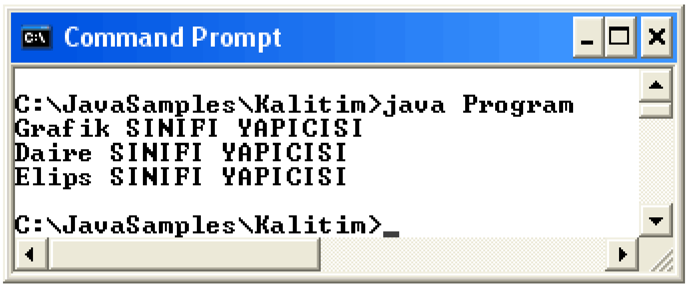
```
    {
        System.out.println("Daire SINIFI YAPICISI");
    }
}
class Elips extends Daire
{
    public Elips()
    {
        System.out.println("Elips SINIFI YAPICISI");
    }
}
public class Program
{
    public static void main(String[] args)
    {
        Elips e=new Elips();
    }
}
```

Örneğin hiyerarşisini daha iyi kavrayabilmek amacıyla kağıt kalemi alıp aşağıdaki gibi grafikleştirmeyide ihmal etmedim.

Burada Elips sınıfı hiyerarşinin el altındaki sınıftır. Grafik sınıfı ise en üstteki sınıftır. Uygulamayı çalıştırdığımda, yapıcı metodların bu hiyerarşik yapıya uygun bir biçimde çalıştırıldığını



gördüm. Yani Elips sınıfından bir nesne türettiğimde, java derleyicisi, bu sınıfın yer aldığı hiyerarşideki en üst sınıfa kadar çıktı ve ilk olarak bu en üstteki sınıfın yani Grafik sınıfının yapıcısını çağırdı. Bu bana, türetilmiş bir nesne yaratıldığında, türetilmiş sınıflar için ortak olan özelliklerin, temel sınıf yapıcısı içerisinden başlangıç ayarlarına getirilebileceğini gösterdi. Böylece her türemiş nesne sayesinde, temel sınıftaki ortak alanların değerlerinide nesne oluşturulurken ayarlayabilirdik.



Varsayılan yapıcılar için geçerli olan bu durum acaba, parametre alan yapıcılar için nasıl işleyecekti? Hiyerarşide üst sınıflara çıkıldıkça, en üst sınıftan aşağıya doğru tüm yapıcıların mutlaka çalışacağı kesindi. Peki değeleri nasıl aktaracatık. Daha net düşündüğümde, C# dilinde yapıcılar için kullandığım base anahtar sözcüğünün yerini java dilinde ne alıcaktı?

Kaynak araştırmalarım, bunun için super bir anahtar sözcüğün olduğunu gösterdi. Super ismindeki bu anahtar sözcük kullanım şekli itibariyle, base anahtar sözcüğünün ilkel haliymiş diyebilirim. Bu durumu incelemek amacıyla aşağıdaki gibi örnek oluşturdum. Burada Grafik sınıfı temel sınıf olarak, Taban ve Yukselik isminde double tipinden değişkenlere sahip. Yapıcı metodunu ise bu alanların değerlerini belirlemek üzere ayarladım. Şimdi gelelim Ucgen sınıfına. Bu sınıfta, Grafik sınıfından türetiliyor. Yapıcı metodu üç parametre almakta. İlk iki parametre, temel sınıf olan Grafik sınıfındaki yapıcılar vasıtasıyla ayarlanabilir. İşte bu amaçla super anahtar kelimesini kullanarak bu iki parametreyi bir üstteki sınıfın yapıcı metoduna gönderiyorum.

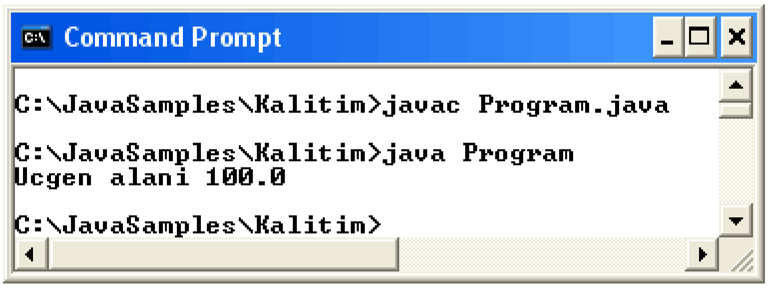
Bu benim ne işime yaradı peki? Uygulamada Ucgen sınıfından bir nesneyi 3 parametre alan yapıcı metodu ile oluşturduğumda, ilk iki parametre, temel sınıftaki yapıcıya aktarılıyor ve böylece benim Grafik sınıfından türettiğim nesnelerin ortak özellikleri olan Taban ve Yüksekliği, türeyen sınıf içinden tekrar bildirmek zorunda kalmıyorum.

```
class Grafik
{
    double Taban;
    double Yukseklik;
    public Grafik(double a,double b)
    {
```

```
        Taban=a;
        Yukseklik=b;
    }
}

class Ucgen extends Grafik
{
    String UcgenTuru;
    public Ucgen(double yc,double yuk,String tip)
    {
        super(yc,yuk);
    UcgenTuru=tip;
    }
    public double Alan()
    {
        return (Taban*Yukseklik)/2;
    }
}
public class Program
{
    public static void main(String[] args)
    {
        Ucgen u=new Ucgen(10,20,"Eskenar");
        System.out.println("Ucgen alani "+u.Alan());
    }
}
```

Örneği çalıştırdığımda aşağıdaki ekran görüntüsünü elde ettim.



Yapıcı metodlar arasındaki parametre aktarımı ile ilgili kafama takılan nokta, super tekniği ile acaba hiyerarşinin en tepesindeki yapıcıyamı gidiliyor sorusuydu. C# dilinde base anahtar kelimesi, bir üstteki sınıfın yapıcılarına gönderme yapıyordu. Bu durum java dilindede böyle olmalıydı. Hemen bir

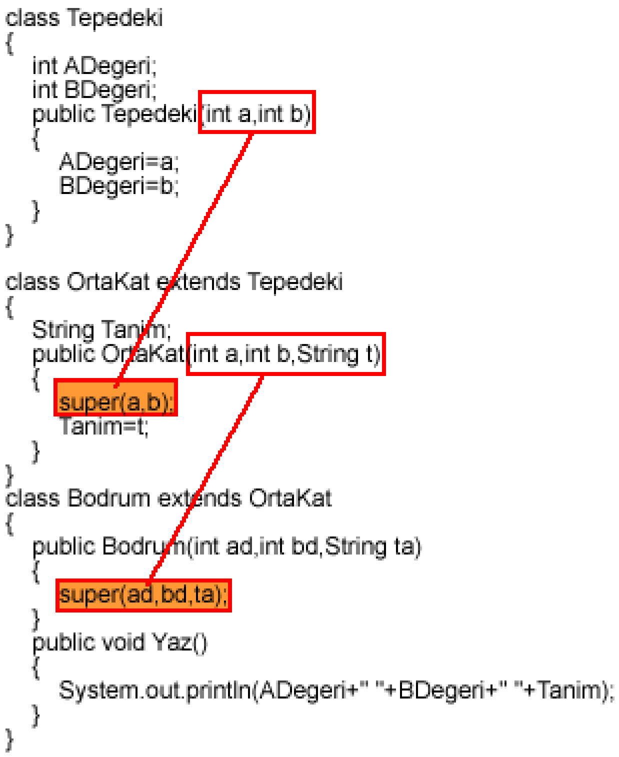
deneme uygulaması yazarak konuyu açıklığa kavuşturmaya çalıştım.

```
class Tepedeki
{
   int ADegeri;
   int BDegeri;
   public Tepedeki(int a,int b)
   {
      ADegeri=a;
      BDegeri=b;
   }
}

class OrtaKat extends Tepedeki
{
   String Tanim;
   public OrtaKat(int a,int b,String t)
   {
      super(a,b);
      Tanim=t;
   }
}
class Bodrum extends OrtaKat
{
   public Bodrum(int ad,int bd,String ta)
   {
      super(ad,bd,ta);
   }
   public void Yaz()
   {
      System.out.println(ADegeri+" "+BDegeri+" "+Tanim);
   }
}
public class Program
{
   public static void main(String[] args)
   {
      Bodrum b=new Bodrum(10,4,"Apratman");
      b.Yaz();
```

```
    }
}
```

Örnekte Bodrum sınıfının yapıcısında kullandığım super anahtar sözcüğü ile bir üst sınıftaki yani OrtaKat sınıfındaki yapıcıya üç parametreyide aktarmış oldum. OrtaKat sınfındaki yapıcı ise gelen parametrelerden ilk ikisini Tepedeki sınfının yapıcısına aktardı. Aslında bu durumu aşağıdaki gibi şekilledirmek anlamak açısından daha kolay olucak.



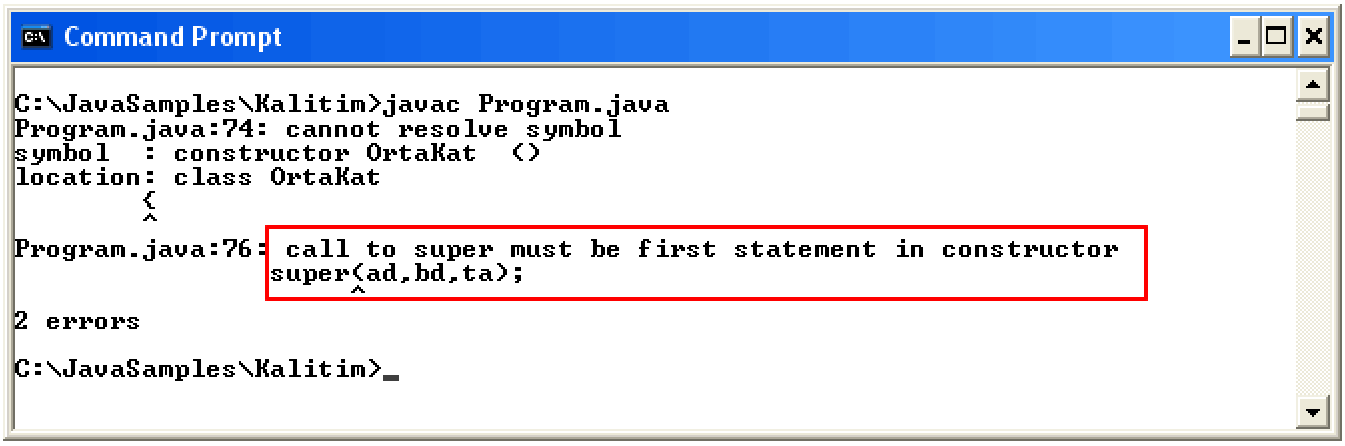
En alttaki sınıf yapıcısından, en üstteki sınıf yapıcısına kadar super tekniğini kullanarak çıkılabilmekteydi. Ancak önemli olan nokta, super tekniğinin, C# dilindeki base anahtar sözcüğünde olduğu gibi, türetilen sınıfın bir üstündeki sınıfa göndermeler yaptığıydı.

Java dilide C# dili gibi kesin askeri kurallar üzerine kurulmuş bir dil. Bu kanıya nerden mi vardım? Yukarıdaki örnekte aşağıdaki gibi bir değişiklik yaptım.

```
class Bodrum extends OrtaKat
{
   public Bodrum(int ad,int bd,String ta)
```

```
    {
       System.out.println("super'den onceki satir");
    super(ad,bd,ta);
    }
    public void Yaz()
    {
       System.out.println(ADegeri+" "+BDegeri+" "+Tanim);
    }
}
```

Tek yaptığım super anahtar sözcüğünün kullanımından önce basit bir kod satırı eklemekti. Ancak sonuçta aşağıdaki hata mesajını aldım. Super, yapıcı metod içerisinde mutlaka ilk satırda kullanılmalıydı.



Kalıtım ile ilgili bir diğer önemli konu ise, C# dilinden isim gizleme (name hidding) olarak bildiğim konunun nasıl ele alındığıydı. C# dilinde, türeyen sınıf ve temel sınıflarda aynı üyeleri tanımladığımızda, türeyen sınıftaki üyenin, temel sınıftaki üyeyi gizlediğini biliyordum. Bu durumun Java dilinde nasıl oldğunu görmek için tek yapmam gereken, temel ve türeyen sınıflarda aynı üyeleri kullanıp, türeyen sınıf nesnesi üzerinden bu üyeye erişmeye çalışmaktı. Bu amaçla aşağıdaki önemsiz, herhangibir işe yaramayan ama bana isim gizlemenin nasıl olduğunu gösterecek kodları yazdım.

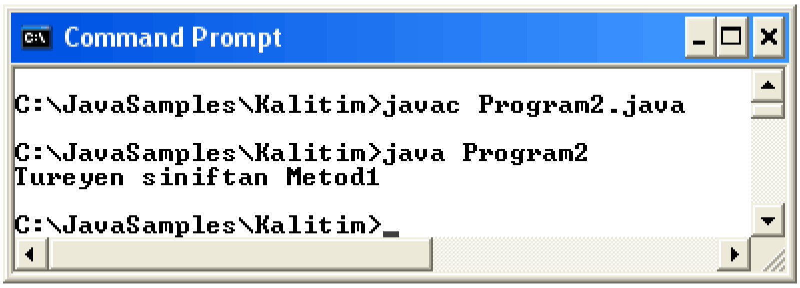
```
class Temel
{
    public void Metod1()
    {
```

```
        System.out.println("Temel sınıftan Metod1");
    }
}

class Tureyen extends Temel
{
    public void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
}

public class Program2
{
    public static void main(String[] args)
    {
        Tureyen t=new Tureyen();
        t.Metod1();
    }
}
```

Bu uygulamayı derleyip çalıştırdığımda aşağıdaki sonucu elde ettim.



C# dilindeki gibi olmuş ve türeyen sınıftaki metod temel sınıftakini gizlemişti. Ancak arada belirgin bir fark vardı. C# dilinde, derleyici kullanıcıyı metod gizlemeye çalıştığı yönünde uyarır ve new operatörünü kullanmamızı ister. Bu aynı zamanda, türeyen sınıftaki üyenin, temel sınıfta aynı isimli başka bir üyeyide gizlediğini açıkça belirtir. Elbetteki bunun bize sağladığı katkı kodun kolay okunabilirliği ve türeyen sınıftaki hangi üyelerin temel sınıf içerisinde aynen yer aldığının bilinmesidir. Bu bana kalırsa Java dilindeki bir eksiklik. C#

dilinde bu eksiklik giderilmiş ve new anahtar sözcüğü işin içine katılmış ki buda bir gerçek.

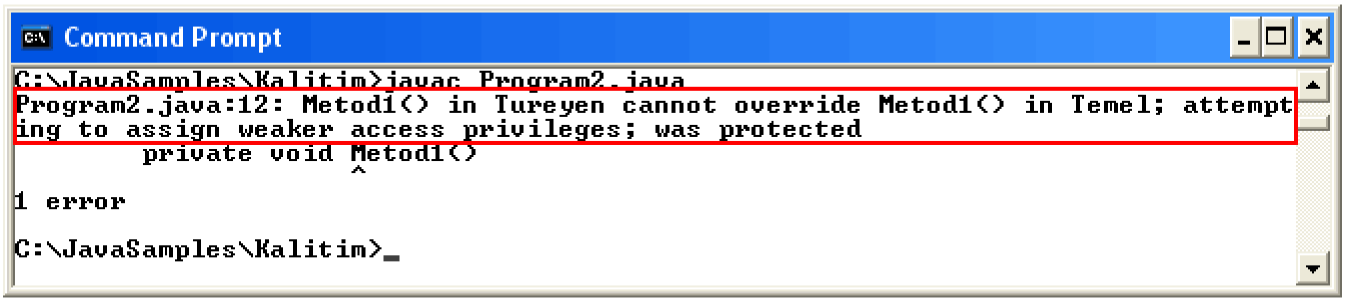
Java dilinde temel sınıf üyelerinin, türeyen sınıfta yeniden bildirilmesi override olarak adlandırılıyor. Yani temel sınıftaki metod türeyen sınıfta geçersiz hale geliyor. Ancak kaynaklardan edindiğim bilgiye göre, bu üyelerin erişim belirleyicilerinin büyük bir önemi var. Şöyleki; aşağıdaki örneği uygulamaya çalıştığımda,

```
class Temel
{
    protected void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
}

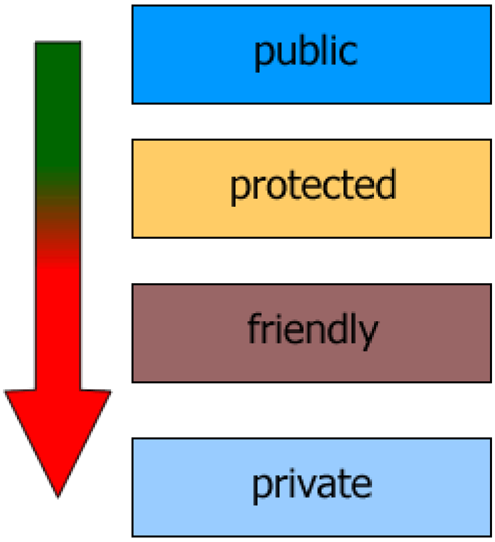
class Tureyen extends Temel
{
    private void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
}

public class Program2
{
    public static void main(String[] args)
    {
        Tureyen tureyen=new Tureyen();
        tureyen.Metod1();
    }
}
```

aşağıdaki hata mesajı ile karşılaşıverdim.



Buradan şu sonuç çıkıyordu. Türeyen sınıfta geçersiz hale getirilen metod, temel sınıftaki metodlar ile ya aynı erişim belirleyicisine sahip olmalı yada daha erişilebilir bir erişim belirleyicisi kullanılmalıydı. Biraz düşündüğümde olayın ciddiyetine vardım. Erişim sözünün bir cümlede bu kadar çok kullanılması biraz sonra kavramsal açıdan kafada bulanıklık yapıcak şeyler açıklanması anlamına geliyordu. Gerçektende eğer türeyen sınıftaki metodu protected veya public yaparsak (ki burada public "daha erişilebilir" manasına geliyormuş) o zaman kodumuz sorunsuz şekilde çalışacaktı. Erişim belirleyicilerinin temel sınıf ve türeyen sınıf arasında oynadığı rolü anlatabilmek için yapılabilecek en güzel şey bu işlemi kafada şekillendirmek ve grafiğe dökmekti.



Şekildeki okun aşağıya doğru inmesinin sebebi, erişim belirleyicileri arasındaki erişilebilirlik sınırlarının durumudur. Public en üst mertebeden bir erişim belirleyicisi olarak her kes tarafında erişilebilirken, en altta yer alan private erişim belirleyicisi en düşük rütbeli erişim belirleyicisidir.

Bu şekile göre düşünüldüğünde şunu söyleyebilirim artık. Temel sınıfta friendly erişim belirleyicisine sahip olan bir üye, türeyen sınıfta ya friendly olmalıdır yada protected veya public olmalıdır. Denemesi bedava deyip kolları sıvadım. Temel sınıftaki metodu friendly yaptım ve ilk olarak türeyen sınıfta aynı erişim belirleyicisini kullandım. Tabi hemen klasik olarak bir dili yeni öğrenmeye başlayan birisi gibi hataya düştüm ve metodların başına hemen friendly erişim belirleyicisini yazıverdim. Her şeyi açıkça belirtecem ya...Oysaki yazmamam zaten bu anlama geliyordu ve Java derleyicim beni uyararak hatamın farkına varmamı sağladı.

```
class Temel
{
    void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
}

class Tureyen extends Temel
{
    void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
}
```

Kod sorunsuz bir şekilde çalışmıştı. Şimdi ise, türeyen sınıftaki metodu friendly erişiminden daha üst kademede olan protected yaptım.

```
class Temel
{
    void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
}
```

```
class Tureyen extends Temel
{
    protected void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
}
```

Kod yine sorunsuz bir şekilde çalıştı. Sırada türeyen sınıftaki metodu en üst seviyede erişilebilirlik sağlıyan public yapmak kalmıştı.

```
class Temel
{
    void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
}

class Tureyen extends Temel
{
    public void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
}
```

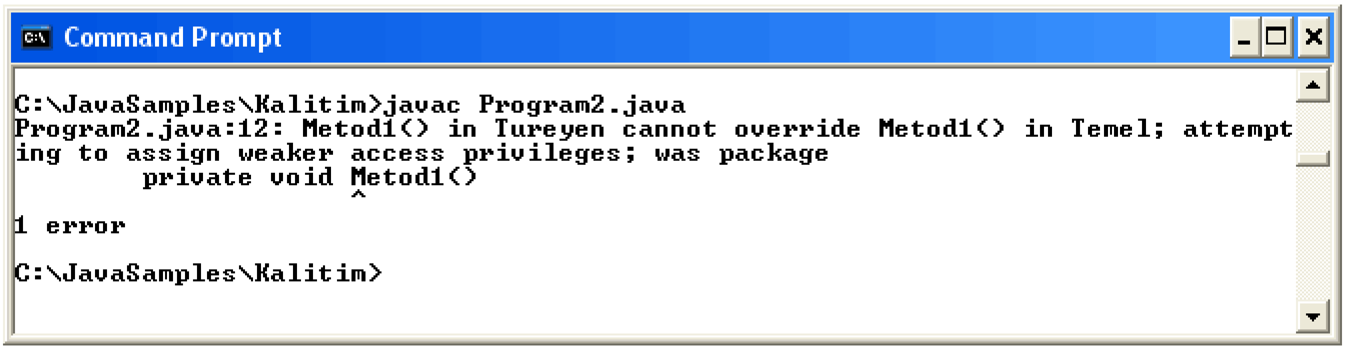
Çok güzel. Bu seferde kod çalıştı. Şimdide durumu ispatlamamı tamamlayacak anti tezi uygulamam gerekiyordu. Yani, türeyen sınıftaki metodu, temel sınıftakinden daha düşük seviyeli bir erişim belirleyici ile (bu durumda bir tek private kalıyor) kullanmaya çalışmak.

```
class Temel
{
    void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
```

```
}

class Tureyen extends Temel
{
   private void Metod1()
   {
      System.out.println("Tureyen sınıftan Metod1");
   }
}
```

Ta taaaa.... Beklediğim hata gerçekleşmişti.



Kalıtım ile ilgili temeller bitmek bilmiyordu. Kaynakları karıştırdıkça düşebileceğim başka tuzaklarda karşıma çıkıyordu. Bunlardan bir tanesi farklı paketler içerisinde yer alan sınıflar arası kalıtım söz konusu olduğunda, türeyen sınıftaki üyelerin geçersiz kılınması sırasında olabilecek olaylardı. Asıl tuzak soru şuydu; Temel sınıfta friendly olarak tanımlanmış bir üye, başka bir pakette bu sınıftan türeyen bir sınıf içinde geçersiz kılınırmıydı?

Denemesi bedeva deyip denemektense, önce akıl muhakemesi yaparak konuya yaklaştım ve bir karara vardım. Friendly erişim belirleyicisi bir üyeye sadece bulunduğu paket içinden erişim hakkı veriyordu. Dolayısıyla başka bir pakette, türetilen bir sınıf içinde bu üye geçersiz kılınamazdı. Herşeyden önce, türeyen sınıf, temel sınıfın bulunduğu paketteki friendly erişim belirleyicilerinden haberdar değildi ve bu aslında durumu açıklıyordu. Türeyen sınıftaki metod, türediği sınıftaki metodtan haberdar değil ise onu nasıl geçersiz kılabilirdiki.

Bu varsayım ışığında, bir örnek ile konuyu derinlemesine incelemeye karar verdim. Yapmam gereklen bir paket içerisinde

yer alan bir sınıftaki friendly metodu, başka bir pakette bu sınıftan türeyen bir sınıf içinde geçersiz kılmaya çalışmaktı. Önce temel sınıfın bulunduğu paketi yazdım.

```
package com.bsenyurt.mat;

public class Temel
{
    void Metod1()
    {
        System.out.println("Temel sınıftan Metod1");
    }
    public int Toplama(int ilksayi,int ikincisayi)
    {
        return ilksayi+ikincisayi;
    }
}
```

Şimdi farklı bir pakette, Temel sınıfından başka bir sınıf türetecek ve Metod1'i buradada kullanacaktım.

```
package com.bsenyurt.yazi;
import com.bsenyurt.mat.*;

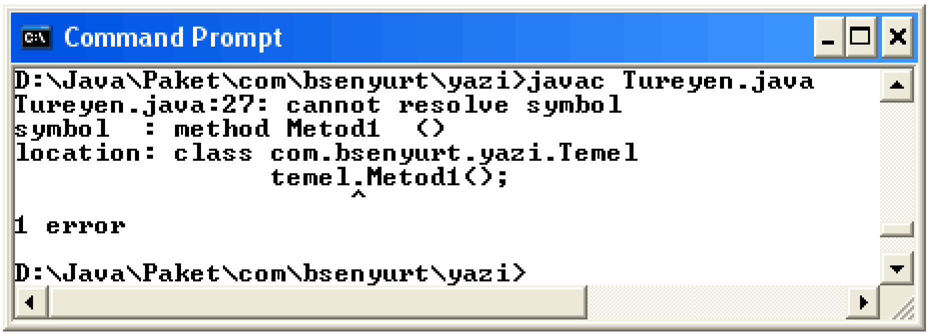
public class Tureyen extends Temel
{
    public void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }

    public static void main(String[] args)
    {
        Tureyen t=new Tureyen();
        t.Metod1();
    }
}
```

Uygulama sorunsuz şekilde çalıştı. Ancak kaynaklarda söylendiği gibi, tureyen sınıfın temel sınıftaki Metod1'in

varlığından haberdar olmadığını nasıl görebilirdim. Sorun buradaydı. Eğer bunu başarabilirsem, zaten türeyen sınıftaki Metod1 in kendi başına bir metod olduğunu yani aslında temel sınıftaki metodu geçersiz kılmadığını söyliyebilirdim. Bu işi iki satırlık kod parçası çözdü. Temel sınıftan bir nesne türetilecek ve bu nesne üzerinden Metod1 çağırılacaktı.

```
Temel temel=new Temel();
temel.Metod1();
```



Gerçektende kaynaklarda söylendiği gibi olmuştu. Temel sınıftan bir nesne oluşturabilmiştim fakat Metod1'e friendly olduğu için erişememiştim. Dahası bu, türeyen sınıfın Temel sınıftaki metotdan haberdar olmadığı anlamına gelmekteydi.

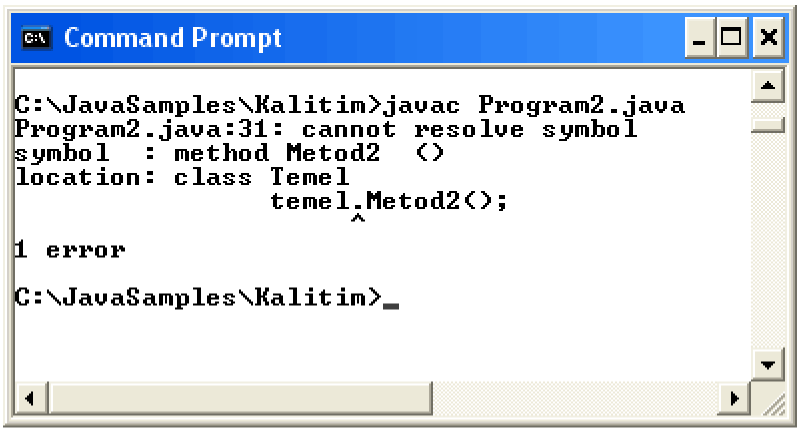
Şimdi ise kalıtım ile ilgili olarak aklıma takılan başka bir konu vardı. Bu c# dilinde sıkça yapılan ve sanal metodların kullanımını doğuran bir testtir. Acaba türeyen sınıf türüden bir nesneyi bir temel sınıf nesnesine aktarsak ve ardından, temel sınıf nesnesi üzerinden, türeyen sınıftaki bir üyeye ulaşmaya çalışsak ne olurdu? C# dilinde bu, derleyicinin hata vermesine neden olmaktaydı. Çünkü temel sınıf türeyen sınıf üyeleri hakkında herhangibir bilgiye sahip olamazdı. Ancak sanal metod tanımalamaları ile, temel sınıf nesnesi üzerinden türeyen sınıftaki bir üyeye erişmek mümkün olabiliyordu. Yani temel sınıftaki sanal metod türeyen sınıfta geçersiz kılınıyordu (override). Acaba java dilinde durum nasıldı? Bunu öğrenmenin yolu her zamanki gibi iyi bir testten geçiyordu.

```
class Temel
{
   public void Metod1()
```

```
    {
        System.out.println("Temel sınıftan Metod1");
    }
}

class Tureyen extends Temel
{
    public void Metod1()
    {
        System.out.println("Tureyen sınıftan Metod1");
    }
    public void Metod2()
    {
        System.out.println("Tureyen sınıftan Metod2");
    }
}

public class Program2
{
    public static void main(String[] args)
    {
        Tureyen tureyen=new Tureyen();
        Temel temel=new Temel();
        temel=tureyen;
        temel.Metod2();
    }
}
```



Beklediğim gibi bir sonuç. Her ne kadar derleyici hatası bana fazla anlamlı gelmesede, Temel sınıf nesnesi üzerinden türeyen

sınıftaki üyeye erişememiştim. Peki ya acaba Java'da bana bu imkanı verecek C# taki sanal metodlar gibi unsurlar varmıydı?

Java dilini öğrenmeye başladığımda bana bu konu ile ilgili pek çok kaynak gerekmişti. Yakın bir arkdaşım bu konuda bana yardımcı oldu ve KaZaAraaa (bilmeden, istemeden, ansızın karşımıza çıkan) bir kaç e-book getirdi. Bu kitaplardan virtual anahtar sözcüğünü arattığımda, sadece JVM (Java Virtual Machine) konularına ulaştım. Amacım C# taki virtual metodları Java dilinde bulabilmekti. Ancak henüz bu muradıma erişemedim. Belkide ve büyük olsalılıkla Java dilinde, virtual metodlar yerine başka elemanlar kullanılıyor olabilirdi. Kim bilir ilerleyen zamanlarda karşıma çıkar umarım. Belki bir sonraki hafta yoğun şekilde inceleyeceğim çok biçimlilik (polimorphsym) konusunda.

Bölüm 8: Final

Yoksa sonuna mı geldim? Olurmu hiç öyle şey canım, daha dünya kadar yolum var. Kaynaklarımdan Java dilini öğrenmeye çalışırken, final anahtar sözcüğünden bahseden yerlere rastladığımda, java dilini daha başlarda bitirmek isteyeceklerini düşünmemiştim zaten. Çoğu kaynakta araya sıkıştırılmıştı bu kavram ancak sağladıkları ile bence öğrenilmesi gereken niteliklerden birisiydi.

Herşeyden önce mükemmelliğin ayrıntılarda gizli olduğunu biliyordum. Gerçektende çoğu kaynak final kavramına sadece yorum satırları ile değinmiş olsada, içerik açısından ayrıca incelenmeyi gerektiriyordu. Aslında buna değerdide. Söz gelimi, geçen hafta üzerinde çalıştığım kalıtım konusuna kattığı bir özellik vardı. Eğer kaynaklarımda bu kavrama rastlamamış olsaydım, bu kabiliyeti bilemiyecek ve karşılaştığımda bu nedir diyerek sağa sola bakacaktım.

O nedenle bu hafta boyunca, final anahtar sözcüğünün Java diline sağladıkların incelemeye çalıştım. Herşeyden önce işe, kalıtım ile başlamam gerektiğini düşünüyordum. final anahtar

kelimesinin kullanılabildiği alanlardan bir tanesi metodlardı. Ancak sadece metodlara değil, değişkenlere, sınıf örneklerine ve sınıflarada uygulanabiliyordu. Bunlarla birlikte kalıtım söz konusu olduğunda final metodlardan bahsetmeden geçilemiyeceğini bu haftaki çalışmalarımdan anlamıştım. İşe öncelike final kelimesinin, metodlara kattığı anlamı tanımlamakla başlamak gerekir diye düşünüyorum.

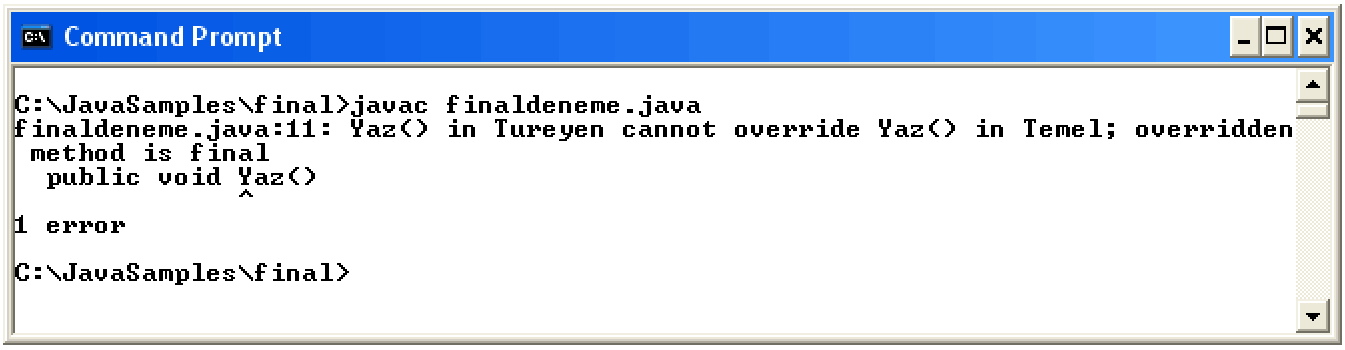
Bir final metod, Temel sınıfta uygulandığında, türetilen sınıflarda iptal edilemeyen metodları belirtir. Yani override işleminden bahsetmek mümkün değildir. Bu etkiyi ispat etmek için her zaman olduğu gibi kahvemden bir yudum aldım ve bir çırpıda aşağıdaki kodları yazıverdim. Tek amacım, final metodların türeyen sınıflarda yeniden yazılıp geçersiz kılınıp kılınamıyacağıydı.

```
class Temel
{
    final void Yaz()
    {
        System.out.println("Temel sınıf Yaz Metodu");
    }
}

class Tureyen extends Temel
{
    public void Yaz()
    {
        System.out.println("Tureyen sınıf Yaz Metodu");
    }
}

public class FinalDeneme
{
    public static void main()
    {
        Tureyen t=new Tureyen();
        t.Yaz();
    }
}
```

Uygulamayı derlediğimde, aşağıdaki derleme zamanı hata mesajı ile karşılaştım.



```
C:\JavaSamples\final>javac finaldeneme.java
finaldeneme.java:11: Yaz() in Tureyen cannot override Yaz() in Temel; overridden
 method is final
  public void Yaz()
              ^
1 error

C:\JavaSamples\final>
```

Her şey sonderece açıktı. Gerçektende temel sınıflarda final olarak tanımlanan metodlar, türeyen sınıflarda override edilemiyorlardı. Final anahtar kelimesinin kabiliyetleri sadece metodlara yaptırdıkları ile sınırlı değildi. Örneğin final alanlar vardı. **Değişkenleri final olarak bildirdiğimizde, onlara ilk değerlerin atanmasından sonra, değerlerini değiştiremiyorduk.** Bu aslında C# dilinde constant değişkenlerden farklı değildi. Yani uygulamada değeri bir kere atandıktan sonra değişmesini istemediğimiz değişkenler için kullanıyorduk.

Final olarak belirlenen değişkenler ile ilgili ilginç olan nokta, değer atamalarının çalışma zamanında (runtime) veya derleme zamanında (debug time) yapılabilmesiydi. Yani istersem, final bir değişkene ilk değer atamasını çalışma zamanında gerçekleştirebilirdim. Ancak çalışma zamanına kadar değeri belli olmayan bir değişken nasıl olabilirdi diyede düşünmeye başlamıştım. Tam o sırada imdadıma Math sınıfının rastgele sayılar üretmekte kullanılan, random metodu yetişti. Bu metod çalışma zamanında, rastgele sayılar üretebilirdi. Dolayısıyla rastgele üretilen bir değeri, programın çalışması boyunca sabit bir değer olarak saklamak için final anahtar kelimesi kullanılabilirdim. Bu bilgiler ışığında, konuyu daha iyi kavrayabilmek için, hemen basit bir örnek geliştirmeye karar verdim.

```
public class FinalAlanlar
{
    final static double deger2;
```

```
    public static void main(String[] args)
    {
        final int deger1=7;
        deger2=0.125;
        final int deger3=(int)(Math.random()*10);

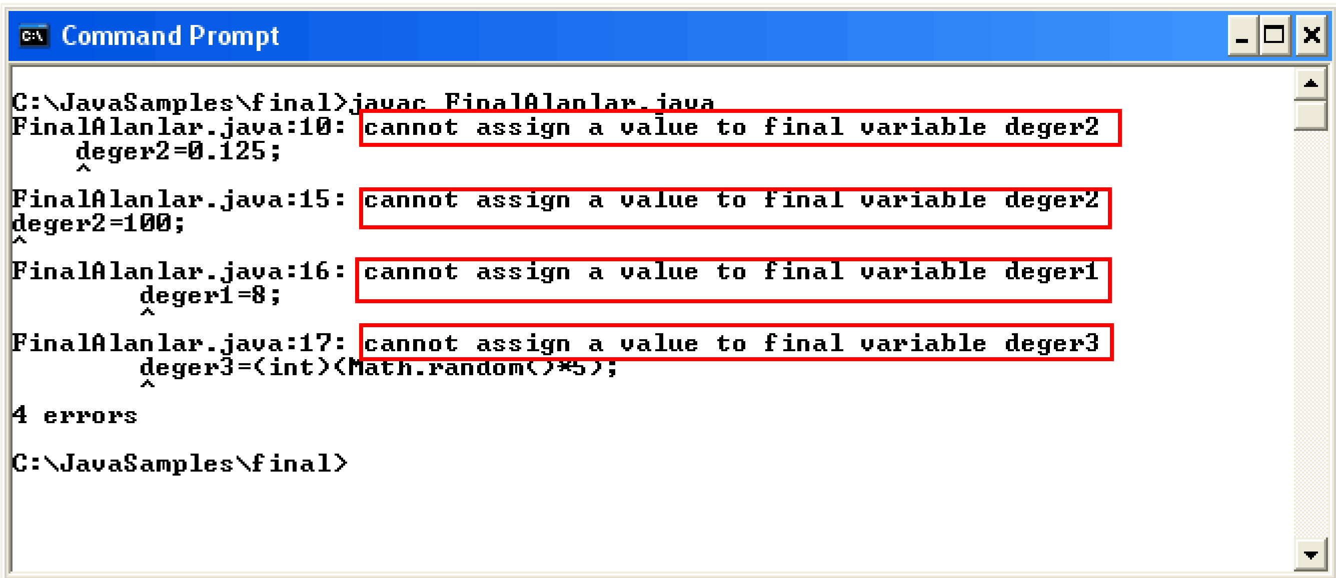
        Yaz(deger1,deger2,deger3);

        deger2=100;
        deger1=8;
        deger3=(int)(Math.random()*5);

        Yaz(deger1,deger2,deger3);
    }

    static void Yaz(int a,double b,int c)
    {
        System.out.println("Deger1="+a);
        System.out.println("Deger2="+b);
        System.out.println("Deger3="+c);
    }
}
```

Uygulamada yapmak istediğim, final olarak belirlenmiş alanların değerlerini değiştirip değiştiremiyeceğimdi. Bu haliyle çalışan uygulama bana 4 adet derleme zamanı hatası verdi.

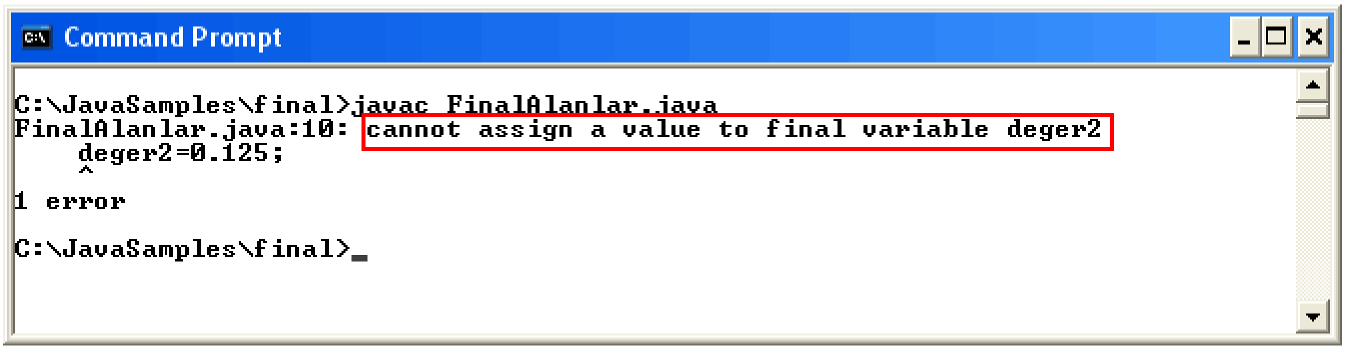


İlk göze çarpan, deger2, deger1 ve deger3 değişkenlerine yeni değerlerin final değişkenler oldukları için atanamıyacağıydı. Diğer taraftan dikkatimi çeken bir başka nokta, ilk değer ataması yapmadığım deger2 değişkeni içinde, sonradan yaptığım ilk değer atamasında hata almamdı. Buradan, final olarak tanımlanan değişkenlere tanımlandıkları anda değer atanması gerektiği sonucuna varıyordum. Ama gerçekten durum böylemiydi? Öncelikle koddaki hataların sayısın indirgemem gerektiğini düşünerek harekete geçtim. Bir yudum sıcak kahve ve ardından aşağıdaki kodları uygulamadan çıkardım.

```
deger2=100;
deger1=8;
deger3=(int)(Math.random()*5);

Yaz(deger1,deger2,deger3);
```

Şimdi uygulamayı yeniden derlediğimde, hata sayısının bire düştüğünü ama deger2 için yapılan atamada halen hata aldığımı gördüm.



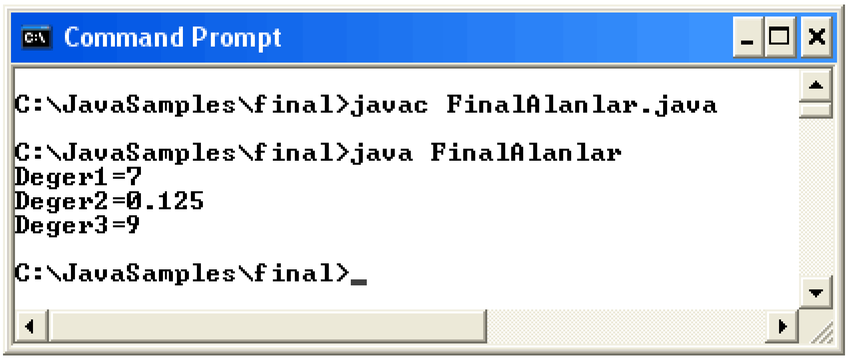
O halde yapılacak tek bir şey vardı, deger2 nin değerini tanımlandığı satırda belirlemek. Kodları son olarak aşağıdaki hale getirdiğimde herhangibir hata mesajı almadan çalıştığını gördüm. Gerçektende final olarak belirlenmiş alanlar programın çalışması esnasında değiştirilemiyorlardı.

```
public class FinalAlanlar
{
    final static double deger2=0.125;
```

```
    public static void Main(String[] args)
    {
        final int deger1=7;
        // deger2=0.125;
        final int deger3=(int)(Math.random()*10);

        Yaz(deger1,deger2,deger3);
    }

    static void Yaz(int a,int b,int c)
    {
        System.out.println("Deger1="+a);
        System.out.println("Deger2="+b);
        System.out.println("Deger3="+c);
    }
}
```



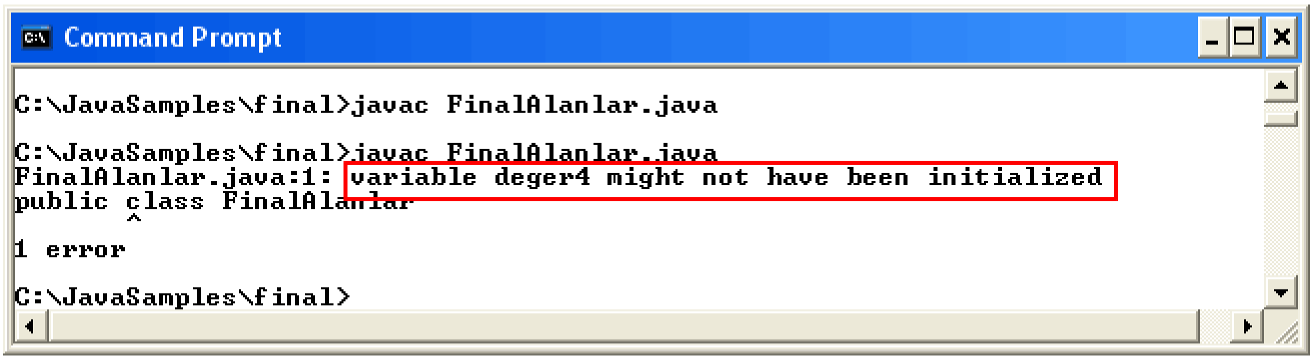
Diğer taraftan aklıma takılan bir nokta vardı. Final olarak belirtilmemiş bir alana ilk değer atamadığımızda java derleyicisi bu değişkenin türüne göre bir ilk değer ataması yapıyordu. Örneğin uygulamada double tipinden bir değişken tanımlayıp buna ilk değer ataması yapmassak derleyici bizim için bu değişkene başlangıç değeri olarak 0.0 değerini atıyordu. Oysa değişkeni final olarak belirttiğimizde, derleyici bizim için varsayılan bir değer atamsı yapmamakta, üstüne üstelik birde hata mesajı vermekteydi. Bunu görmek için uygulamaya deger4 isminde double veri türünde bir değişken ekledim. Ancak herhanbiri değer atamadım.

```
static double deger4;
```

Daha sonra, bu degeri ekrana yazdırdığımda derleyicinin 0.0 değerini atadığını gördüm. Ancak, bu değişkeni final olarak belirttiğimde,

```
final static double deger4;
```

ve programı derlediğimde aşağıdaki hata mesajını aldım.



Elde edilen bu hata mesajı bana, final olarak tanımladığım değişkenlere mutlaka ilk değer ataması yapmam gerektiğini ve hatta bu atamayı değişkenleri tanımladığım yerde yapmam gerektiğini gösterdi.

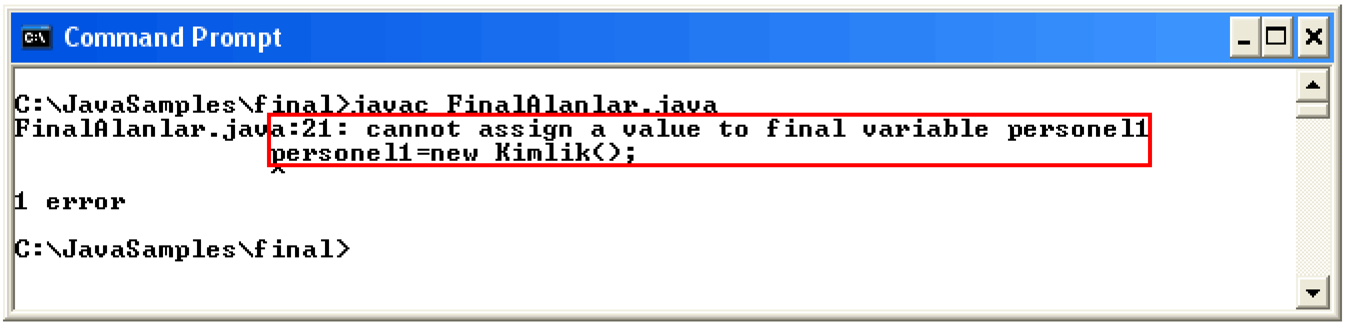
Final anahtar kelimesinin değişkenlerce kullanılması bu şekildeyken, pandora'nın kutusu açılmaya devam ediyor ve içinden sınıf nesne örnekleri çıkıyordu. **Final anahtar kelimesini , nesne örneklerinede uygulayabilir ve böylece bu nesnelerin programın çalışması boyunca sadece bir kez yaratılmasını sağlayabilirdik.** Bu durumu araştırmak amacıyla aşağıdaki kodları yazdım. Burada tanımladığım bir sınıfa ait nesneyi final anahtar kelimesi kullanarak oluşturdum ve daha sonra aynı nesneyi tekrardan oluşturmaya çalıştım.

```
class Kimlik
{
    String ad="Burak";
    String soyad="SENYURT";
    int id=1000;

    public void Yaz()
    {
        System.out.println(ad+" "+soyad);
```

```
    }
}
public class FinalAlanlar
{
    public static void main(String[] args)
    {
        final Kimlik personel1=new Kimlik();
        personel1.Yaz();
        personel1=new Kimlik();
    }
}
```

Uygulamayı derlediğimde aşağıdaki hata mesajını aldım.

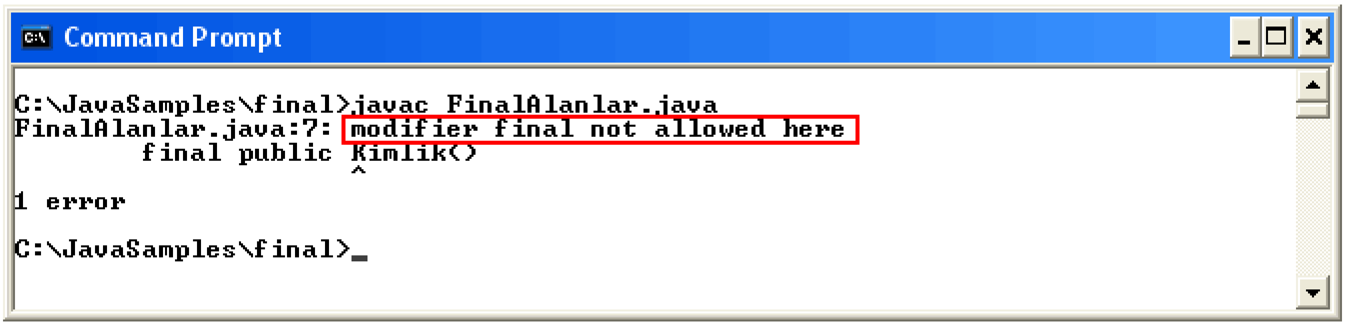


Sonuç olarak, final olarak oluşturulan bir sınıf örneği bir kere oluşturulabiliyordu. Diğer yandan final olarak tanımlanan bu sınıfın üyelerine normal olarak erişilebiliyordu. Final anahtar kelimesinin, bir sınıf örneğinin sadece bir kere üretilmesini sağlaması benim aklıma başka bir şey getirmişti. Acaba bir sınıfın yapıcı metodlarına final anahtarını ekleyebilirmiydim? Nede olsa final anatar kelimesini sınıf metodları ile birlikte kullanabiliyorduk. Bunun anlamanın yolu denemekten geçiyordu. Önce aşağıdaki varsayılan yapıcıyı ekleyerek işe başladım.

```
final public Kimlik()
{

}
```

Programı derlediğimde aşağıdaki hata mesajını aldım.



Belkide, final anahtar sözcüğünün yerinden kaynaklanıyordur diye düşündüm. Bu sefer final anahtar kelimesini public tanımından sonraya aldım. Ancak yine aynı hata mesajını elde ettim. Aklıma bu kez parametre alan bir yapıcıda final anahtarını kullanmak geldi. Nitekim yukarıdaki durum belkide varsayılan yapıcılara hastı.

```
final public Kimlik(int YeniID)
{
    id=YeniID;
}
```

Ancak sonuç yine değişmedi. Demekki final metodlar tanımlayabiliyor ancak final yapıcılar oluşturamıyordum. Tabi kendi kendime neden final yapıcılar oluşturmaya çalışayım ki dedim. Biraz anlamsız geldi ama neleri yapamıyacağımı görmem açısındanda bir yerde iyi oldu. Final anahtar sözcüğünü metodlara uygulayarak kalıtımda override'ları engellemeyi, değişkenlere uygulayarak program boyunca değiştirilemeyen değişkenler elde etmeyi, sınıf nesne örneklerine uygulayarak aynı nesnenin tekrar türetilememisini sağlamayı, öğrendikten sonra dahada fazlasının olabileceğini hiç düşünmemiştim.

Meğerse sırada parametreler varmış. **Bir metodun parametrelerini final olarak tanımlayabilir ve bu sayede metodlara gelen parametre değerlerinin değiştirilmesini engelliyebilirdim**. Nasıl mı?

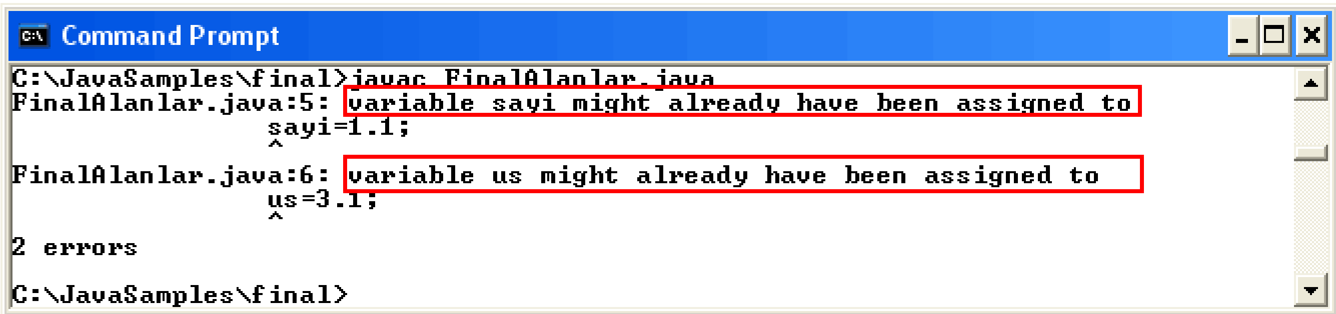
```
class Hesaplar
{
   double UsAl(final double sayi,final double us)
   {
      return Math.pow(sayi,us);
```

```
    }
}
public class FinalAlanlar
{
    public static void main(String[] args)
    {
        Hesaplar islem1=new Hesaplar();
        System.out.println(islem1.UsAl(2.4,1.25));
    }
}
```

Bu uygulamada, Math sınıfının pow metodunu kullanarak, double tipinden bir sayının double değerden üssünü aldım. Önemli olan kısım UsAl metodunda, parametre değerlerinin final olarak belirlenmesiydi. Şimdi bu metod içerisinde, gelen parametre değerlerini değiştirmeye çalışmalıydım.

```
double UsAl(final double sayi,final double us)
{
    sayi=1.1;
    us=3.1;
    return Math.pow(sayi,us);
}
```

Uygulamayı bu hali ile derleyip çalıştırdığımda,



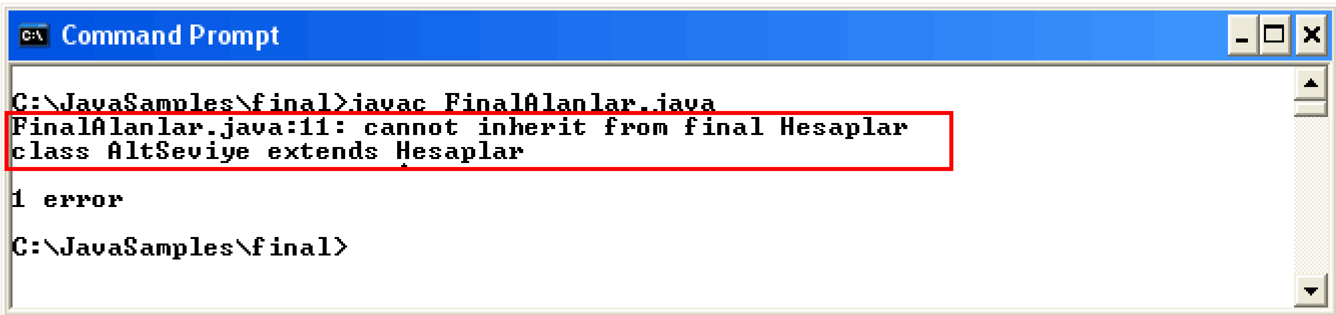
görülen hata mesajını elde ettim. Demekki, metod parametreleri, final olarak bildirilirse, metod içinde değerleri değiştirilemiyordu. Ancak parametreye gönderilen değerlerin verildiği yerlerde bu durum söz konusu değildi. Bu tip parametreye sahip metodları, değerlerin kesinlikle içeride değiştirilmemesini istediğimiz durumlarda kullanabilirdim.

Herhalde final ile ilgili pek çok şey hallolmuştu demelimiydim acaba diye düşünürken, kitaplarımın birisinde final sınıflar isimli bir başlık görüverdim. Evet sanırım daha herşey bitmemişti. Kahvem bitmişti ancak final anahtar kelimesinin işlevsellikleri halen daha bitmemişti. Yinede final sınıf kavramı faydalı ve incelenmeye değer bir kavramdı. Final olarak belirtilen sınıfların en önemli özelliği, C# dilindeki sealed anahtar sözcüğünün işlevselliğine sahip olmalarıdır. Yani **final olarak tanımlanmış bir sınıftan başka bir sınıfı kalıtımsal olarak türetemeyiz.**

```
final class Hesaplar
{
    double UsAl(final double sayi,final double us)
    {
        sayi=1.1;
        us=3.1;
        return Math.pow(sayi,us);
    }
}
class AltSeviye extends Hesaplar
{

}
```

Buradaki basit kodlarda, Hesaplar isimli sınıfı final olarak tanımladım. Sonrada, bu sınıftan kalıtım yolu ile başka bir sınıf türetmeye çalıştım. Elbette sonuç olarak java derleyicisi beni uyardı ve final olarak tanımlanmış bir sınıftan başka bir sınıfı türetemiyeceğimi söyledi.



Sınıfların final olarak tanımlanması, içerdiği metodların veya değişkenlerin final olarak algılanması anlamına gelmiyordu tabiki. Sadece ve sadece sınfın türetme amacı ile

kullanılamıyacağını belirtiyordu. Final anahtar kelimesi ile ilgili elde ettiğim bilgiler şu an için bunlarla sınırlıydı. İlerleyen zamanlarda farklı şekillerde kullanımını görebilirmiyim bilemiyorum. Nitekim bende final olmuş durumdayım. Hem kahvem bitti hemde pilim. Sanıyorumki bir ara vermenin zamanı geldi. Artık önümüzdeki hafta boyunca, polimorfizm konusunu incelemek için bana gerekli olan enerjiyi depolamam gerekiyor.

Bölüm 9: Bukalemun

Bu hafta boyunca, yakınlardaki ev hayvanları dükkanında bulunan bir bukalemun'u inceledim durdum dersem, herhalde bu adam sonunda çıldırdı diyeceksiniz. Yok henüz çıldırmadım. Aslında bu hafta boyunca nesne yönelimli programlama dillerinin temel kavramlarından birisi olan, çok biçimliliği inceledim. Ben çok biçimliliği bukalmenun'lara benzettiğim içinde, sanırım bu dükkanda bir süre oyalandım. Belkide bukalemun'dan bana çok biçimliliğin ne olduğunu söylemesini bekliyordum dersem herhalde gerçekten çıldırdı bu adam diyeceksiniz. Yok henüz çıldırmadım :)

Nasıl ki bukalemunlar, bulundukları ortamın şartlarına uyuyor ve çok değişik kılıklara yada biçimlere bürünebiliyorlarsa, nesne yönelimli bir programlama dilindede bir nesne, bukalemunla aynı tarz davranışları gösterebilmelidir. Bu nesne yönelimli programlama dillerinin doğasında olan önemli bir özelliktir. Ancak, çok biçimlilik, kavram olarak tek başına var olmasına rağmen, nesne yönelimli programlama dili tekniğinde, kalıtımın bir sonucu ve bir parçası olarak karşımıza çıkmaktadır. Gerçektende, çok biçimlilik ve kalıtım içi içe geçmiş iki önemli, nesne yönelimli programlama kavramıdır.

Normalde sınıf kavramı, arkasından gelen kalıtım ve bunlara bağlı olarak ortaya çıkan çok biçimlilik ile, kahveleri içtikçe java dilinin nesne yönelimli temellerinide iyice anlamaya başlamış olduğumu düşünüyorum. Ancak bu kavramlar genelde soyut nitelikler olduğundan tanımlar ile anlaşılmaları gerçekten zor oluyor. İşte ben bu gibi durumlarda, her zaman basit ve aptalca düşünmeye çalışarak, olayı en basit hali ile ele alabileceğim

örnekler geliştirmeye çalışırım. Aynen bu kahve molası için, incelemeyi düşündüğüm çok biçimlilik kavramında olduğu gibi.

Ancak yinede ilk örneğime başlamadan önce, kafamda basit ama etkili bir çok biçimlilik tanımı oluşturmam gerektiği düşüncesindeyim. Bu amaçla şu tanımın çok biçimliliğe uygun olduğu kanısındayım. "**Çok biçimlilik bir nesnenin davranış şekillerinin duruma göre değiştirilebilmesidir."** Aynen, bulunduğu ortamın şartılarına göre renklerini mükemmel bir biçimde ayarlayan sevimli bukalemun bugi gibi. Bugi bu işi gerçekten çok iyi başarıyor. Ancak benim benzer işlevselliği bir sınıf nesne örneğine yükleyebilmem için bilmem gereken yada sahip olmam gereken bazı temel elementler var. Kalıtım, override ve late binding.Kalıtımı önceki kafein molalarında öğrendiğime göre ve override etmeyi bildiğime göre doğrudan çok biçimliliğe geçebilirim sanıyorumki.

Artık kolları sıvayıp bu kavrama girişimi sağlayacak ilk örneğimi geliştirmemin zamanı geldi. Bir yudum kahvenin ardından, aşağıdaki kodları oluşturdum. Burada kalıtım ilişkisi olan sınıf tanımlamaları ve temel sınıfta tanımlanıp türeyen sınıflarda override edilen basit metodlar mevcut. Amacım çok biçimliliği temel sınıf türünden bir nesneye uygulamak. Uygulamak ama bu nasıl olucak ve ne işe yarıyacak? Asıl cevaplanması gereken sorular işte bunlar. Öncelikle tabiki örneği yazıp başarılı bir şekilde derlemem gerekiyor.

```
class TemelSinif
{
   public void Yaz()
   {
      System.out.println("Ben TEMEL sinifim");
   }
}

class Tureyen_1 extends TemelSinif
{
   public void Yaz()
   {
      System.out.println("Ben Tureyen_1 sinifiyim");
   }
```

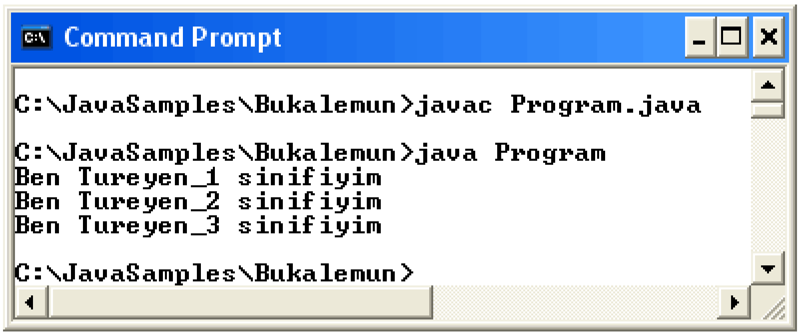
```
}

class Tureyen_2 extends TemelSinif
{
    public void Yaz()
    {
        System.out.println("Ben Tureyen_2 sinifiyim");
    }
}

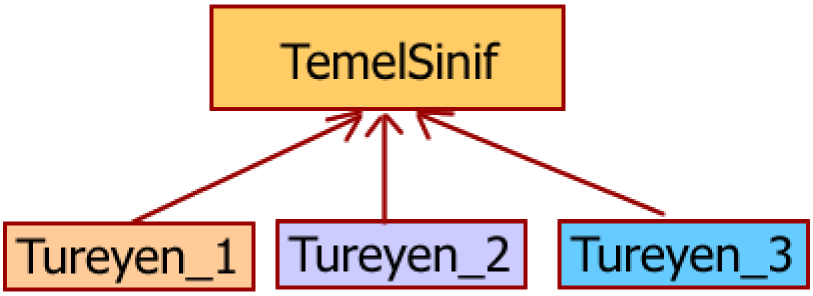
class Tureyen_3 extends TemelSinif
{
    public void Yaz()
    {
        System.out.println("Ben Tureyen_3 sinifiyim");
    }
}

public class Program
{
    public static void Yaz(TemelSinif t)
    {
        t.Yaz();
    }
    public static void main(String[] args)
    {
        Tureyen_1 t1=new Tureyen_1();
        Tureyen_2 t2=new Tureyen_2();
        Tureyen_3 t3=new Tureyen_3();
        Yaz(t1);
        Yaz(t2);
        Yaz(t3);
    }
}
```

Kodu Program.java ismli ile kaydedip derlediğim ve çalıştırdığımda aşağıdaki sonucu elde ettim.



Çalışma şeklini görünce "Oha Falan Oldum Yani" dedim kendimce. Evet bir kod yazmıştım ancak çok biçimlilik bu kodun neresindeydi. Dahası sevimli bukalemun bugi nerelerdeydi; O kadar iyi kamufle olmuştuki koda ilk baktığımda onu görememiştim. Öncelikle kodun içerisinde yer alan sınıfların hiyerarşisine yakından bakmak gerekiyordu. Bu amaçla şu meşhur uml diagramları tekniğini kullanmaya çalıştım ve sınıflar arası ilişkiyi aşağıdaki şekilde olduğu gibi kağıda döktüm.



TemelSinif isimli base class'tan türeyen üç adet derived class'ım vardı elimde. Ayrıca Temel sınıfta tanımlanan Yaz isimli metod türeyen sınıflar içerisinde iptal edilerek yeniden yazılıyordu (override). Buraya kadar her şey gayet açıktı. Bu noktada benim için önemli olan kilit nokta Program sınıfı içerisindeki Yaz metoduydu.

```
public static void Yaz(TemelSinif t)
{
   t.Yaz();
}
```

Bu metodun ilginç olan yanı, TemelSinif türünden bir nesne örneğini parametre olarak alması ve aldığı nesneye ait Yaz metodunu çağırmasıydı. İşte şu kafaları karıştıran çok biçimlilik buydu ve bizim bugi'de çok güzel şekil değiştirerek t nesnesi

oluvermişti. Program sınıfının main metodundaki kod satırları çok biçimliliğin kullanılmasının bir yolunu göstermekteydi.

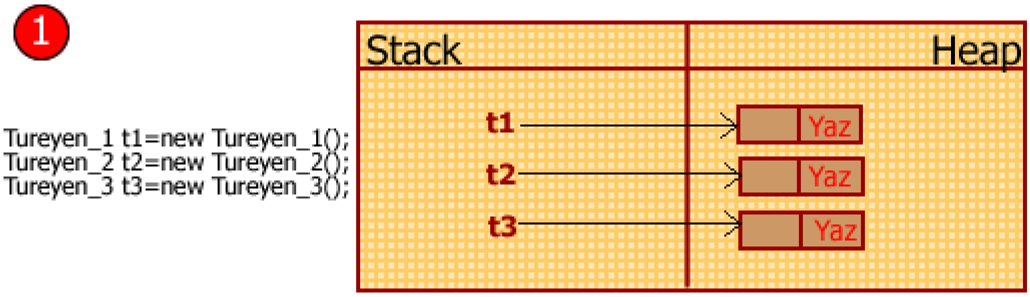
```
public static void main(String[] args)
{
   Tureyen_1 t1=new Tureyen_1();
   Tureyen_2 t2=new Tureyen_2();
   Tureyen_3 t3=new Tureyen_3();
   Yaz(t1);
   Yaz(t2);
   Yaz(t3);
}
```

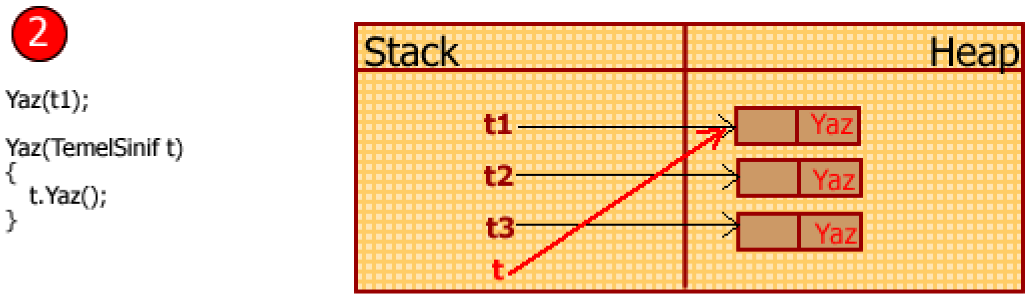
Main metodunda türeyen sınıflardan 3 farklı nesne tanımlanmıştı. Her bir nesnenin Yaz metodunu ayrı ayrı çağırabilirdim. Ancak bunun yerine, tüm bu sınıfların türediği, temel sınıf türünden bir nesne örneğini parametre alan bir metodu çağırmayı ve buradaki türeyen sınıf nesnelerini bu metoda göndermeyi tercih etmiştim. Peki olan olay tam olarak neydi? Örneğin Yaz(t1) ile ne gibi işlemler gerçekleştiriliyordu?

Yaz(t1), Program sınıfım içerisindeki static Yaz metodunu çağırıyor ve buna t1 nesnesini parametre olarak gönderiyordu. Yaz metodunun parametresi ise t1 nesnesinin türetildiği TemelSinif türündendi. Bingo. İşte çok biçimliliğin en temel noktasını yakalamıştım. Türeyen nesne ne olursa olsun, Yaz metodu sadece tek bir nesne kullanıyor ve bu tek nesne üzerinden, kendisine atanan türeyen sınıfa ait üyeleri çalıştırıyordu. Dolayısıyla t1 nesnesi Yaz metoduna aktarıldığında, TemelSinif türünden t nesnesine atanmış ve doğal olarak t.Yaz() kod satırı ile t1'in Yaz metodu çağırılmıştı. Bunu sağlayan elbetteki TemelSinif'in türeyen sınıf üyelerine erişebiliyor olmasıydı.

Çok biçimlilik burada Program sınıfının static Yaz metodunun gerçekleştirdiği işlevsellikti. Hangi sınıfın Yaz metodunun çalıştırılacağı burada belli oluyor dolayısıyla minik bugi t nesne örneği, şartlara uygun hareket ediyordu. İşin suyunu çıkartıp programın çalışma biçimini grafiğe dökmeye karar verdim. Sonuç olarak ikinci bir kahve fincanı doldurmam gerekmişti ama

kafamdaki şekillerde çok biçimsel anlamlar kazanmaya başlamıştı.





Şekilde görüldüğü gibi Program sınıfındaki static Yaz metodu, çalışma zamanında çağırıldığında, TemelSinif türünden bir t nesnesi tanımlanıyor ve bu t nesnesine, Yaz metodunun çağırılmasında gönderilen türeyen sınıf nesnesi atanıyor. Bunun sonucu olarak bu t nesnesi, gelen sınıfın heap bölgesindeki adresini referans etmeye başlıyor. Doğal olarak, t.Yaz metodu ilede,parametre olarak gelen türeyen sınıf nesnesinin Yaz metodu çağırılmış oluyor.

Burada aslında önemli bir nesne yönelimli programlama tekniği kavramıda mevcut. Late Binding(Geç Bağlama). Yukarıdaki şeklin bana söylediği en önemli şey olayların çalışma zamanında gerçekleşiyor olması. Bunun ifade ettiği şey ise oldukça önemli. Nitekim, program çalışmaya başladıktan sonra, biz static Yaz metodunu çağırdığımızda, bu metod içinden hangi türeyen sınıfın Yaz metodunun çağırılacağı henüz bilinmemektedir. Bu ancak, türeyen sınıf nesnesi metoda parametre olarak atanıp, TemelSinif türünden t nesnesine atandığında belli olucaktır. Yani hangi Yaz metodunun çağırılıacağına çalışma zamanında karar verilmektedir. İşte bu olay, geç bağlama (late binding) olarak adlandırılmakta olup, çok biçimliliğin bir parçası olarak, nesne yönelimli programlama dilinin temel yapı taşlarından birisi olarak karşımıza çıkmaktadır.

Çok biçimliliği böylece tanımaya başlamış oldum. Peki bu kavram benim nerede işime yarayabilirdi. Bu cevaplanması gerçekten çok zor bir soru. Bence esas olan çok biçimliliğin bize sağlamış olduğu faydayı anlamaya çalışmak. Ancak yukarıdaki örneği normal programlama teknikleri ile yapmaya çalışsaydık acaba nasıl bir kod yazımı olurdu. Yani çok biçimlilik uygulamadan. Bu noktada uzun zamandır nesne yönelimli diller ile uğraşan birisi olarak bu çeşit bir kodu yazmak benim için oldukça zor oldu. Sonunda aşağıdaki gibi bir sonuç ortaya çıktı. Şekilse açıdan en önemlisi çalışma şekli açısından oldukça kötü bir örnek.

```
class TemelSinif
{
   public void Yaz()
   {
      System.out.println("Ben TEMEL sinifim");
   }
}

class Tureyen_1 extends TemelSinif
{
   public void Yaz()
   {
      System.out.println("Ben Tureyen_1 sinifiyim");
   }
}

class Tureyen_2 extends TemelSinif
{
   public void Yaz()
   {
      System.out.println("Ben Tureyen_2 sinifiyim");
   }
}

class Tureyen_3 extends TemelSinif
{
   public void Yaz()
   {
```

```
        System.out.println("Ben Tureyen_3 sinifiyim");
    }
}

public class Program4
{
    public static void Yaz(Object obj)
    {
        if(obj instanceof Tureyen_1)
        {
            Tureyen_1 t1=(Tureyen_1)obj;
            t1.Yaz();
        }
        else if(obj instanceof Tureyen_2)
        {
            Tureyen_2 t2=(Tureyen_2)obj;
            t2.Yaz();
        }
        else if(obj instanceof Tureyen_3)
        {
            Tureyen_3 t3=(Tureyen_3)obj;
            t3.Yaz();
        }
        else if(obj instanceof TemelSinif)
        {
            TemelSinif t=(TemelSinif)obj;
            t.Yaz();
        }
    }

    public static void main(String[] args)
    {
        Tureyen_1 t1=new Tureyen_1();
        Tureyen_2 t2=new Tureyen_2();
        Tureyen_3 t3=new Tureyen_3();
        Yaz(t1);
        Yaz(t2);
        Yaz(t3);
    }
}
```

Örnekte çok biçimliliğin olmadığını varsayarak hareket ettim. Bu durumda, ortak static Yaz metoduna gönderdiğim nesnelerin tipleri belli olmadığı için, Object sınıfından bir nesne örneğini parametre olarak belirtmem gerekti. Daha sonra bu metoda gönderilen nesnelerin hangi sınıf tipinden olduklarını öğrenmek amacıyla instanceof anahtar kelimesinin kullanıldığı if bloklarını işin içine kattım. Bu if bloklarında nesnenin hangi tipten olduğuna baklıyor ve daha sonra bu nesne için yeni bir örnek new operatörü ile oluşturuluyor ve daha sonra bu yeni nesne örneği üzerinden gerekli Yaz metodu çalıştırılıyordu. Oysaki aynı örneği çok biçimlilik yolu ile ne kadar basit ve anlamlı yapmıştık. Zaten kim söylemiş ise doğru söylemiş. "Basit ama aptalca düşünmeye devam et."

Dolayısıyla sonuç olarak çok biçimliliğin faydası hakkında şunu söyleyebilirim artık; "Çok biçimlilik sayesinde, çalışma zamanında nesne davranışlarına çeşitlilik katabilmekteyiz."

Çok biçimliliği başka nasıl kullanabiliriz diye düşünmeye başladığım sırada aklıma fabrika çalışanları geliverdi. Bir fabrikanın çalışanlarına ilişkin bilgileri tutan temel bir sınıf olduğunu düşündüm. Bu sıfıntan türeyen ve işçiler arasındaki kategorileri birer sınıf olarak temsil eden türemiş sınıflar olucaktı. Temel sınıfta tanımlanmış, saat ücreti üzerinden maaş hesabı yapan bir metodu, türeyen sınıflarda yeniden yazdım yani override ettim. Her türeyen sınıfın kendisine göre bir saat ücreti olacağından, maaş hesaplamalarıda farklı olucaktı. Programın içerisinde, tüm fabrika işçilerinin, temel sınıftan bir dizi içerisinde tutulduğunu düşündüm, bu diziye çok biçimliliği uygulayarak, tek bir nesne üzerinden, hangi türeyen sınıfa ait metodun çalıştırılacağınının çalışma zamanında karara bağlanmasını sağlamaya çalıştım. O halde ne duruyorum. Bu örneği geliştirip çok biçimlilikle iligli bilgileri pekiştirmeye karar verdim ve kahvemden bir fırt çekip uygulamayı yazmaya başladım.

```
class Isciler
{
    public int hesapla()
    {
        return -1;
```

```
    }
}

class Kidemli_Isciler extends Isciler
{
    int saatUcreti=100;
    int saat;

    public Kidemli_Isciler(int s)
    {
        saat=s;
    }
    public int hesapla()
    {
        return saatUcreti*saat;
    }
}

class Vasifli_Isciler extends Isciler
{
    int saatUcreti=80;
    int saat;

    public Vasifli_Isciler(int s)
    {
        saat=s;
    }

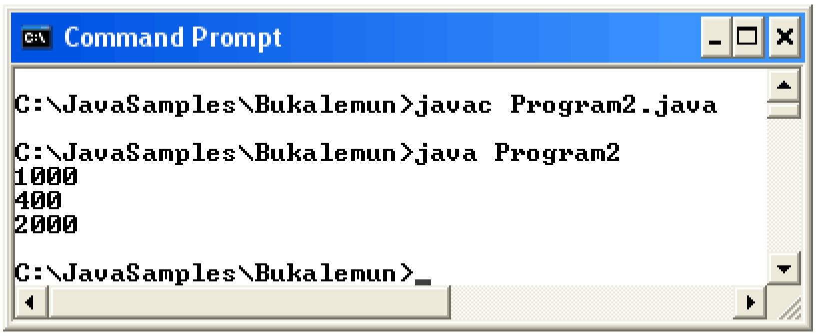
    public int hesapla()
    {
        return saatUcreti*saat;
    }
}

public class Program2
{
    public static void main(String[] args)
    {

        Isciler[] isciler=new Isciler[3];
```

```
        iciler[0]=new Kidemli_Isciler(10);
        iciler[1]=new Vasifli_Isciler(5);
        iciler[2]=new Kidemli_Isciler(20);

        for(int i=0;i<isciler.length;i++)
        {
            int ucret=isciler[i].hesapla();
            System.out.println(ucret);
        }
    }
}
```



Çok biçimlilik ile ilgili gözden kaçan bir ayrıntıyıda çalışmalarımı yaparken farkettim. Çok biçimliliğin uygulanması aslında programların çalışma performansı üzerinde azaltıcı bir etki yaratmaktaydı. Dolayısıyla geç bağlama söz konusu olduğunda uygulamanın verimi düşüyordu. Bu amaçla ilk yazdığım işe yaramayan ancak çok biçimliliğin ne olduğunu gösteren minik program parçasının dahada aptallaştırılmış sürümünü göz önüne aldım.

```
class TemelSinif
{
    public void Yaz()
    {
        System.out.println("Ben TEMEL sinifim");
    }
}

class Tureyen_1 extends TemelSinif
{
```

```
    public void Yaz()
    {
        System.out.println("Ben Tureyen_1 sinifiyim");
    }
}

public class Program3
{
    public static void Yaz(TemelSinif t)
    {
        t.Yaz();
    }

    public static void main(String[] args)
    {
        Tureyen_1 t1=new Tureyen_1();
        TemelSinif t=new TemelSinif();
        Yaz(t1);
        Yaz(t);
    }
}
```

Burada verimi düşeren neydi acaba? Biraz düşününce aslında bunun sebebi anlaşılabiliyordu. Olay Yaz(t1) çağırımında kendisini gösteriyordu. Yaz(t1) ile, static Yaz metodu çağırlıyor TemelSinif nesnesi türünden t sınıfına ait Yaz metodu çalıştırılıyordu. İşte çalışma zamanında java sanal makinesi, bu noktada geç bağlama olup olmadığını kontrol etmekteydi. Burada TemelSinif t nesne örneğine Türeyen_1 sınıfından bir nesne atanmıştı. Dolayısıyla burada geç bağlamadan söz edebilirdik. Bunu gören JVM bu işlemin ardından türeyen sınıfın Yaz metodunun override edilip edilmediğine bakıcak ve ondan sonra bu metodu çağırıcaktı. İşte bu zincir çalışma verimini düşüren bir etken olarakta karşımıza çıkıyordu.

Çok biçimlilik (Polimorphism) kavramı nesne yönelimli programlama kavramlarının önemli bir elemanı olarak artık kafamda iyice şekillenmişti. Artık Java'da ilerlemeye devam edebilirdim. Ama önce alışverişe çıkıp kahve ve süt tozu stoklarımı yenilemem gerekiyor.

Bölüm 10: Soyutlama (Abstraction)

Geçen hafta boyunca soyutlama kavramın irdelemeye çalıştım. Bunu yaparken pek zorlanmadım çünkü C# dilinden soyutlamanın ne olduğunu biliyordum. Ancak bu kavram zaman zaman kafa karıştırıcı bir hal alabilmekteydi. En büyük sıkıntı soyutlamaların tam olarak nasıl ifade edildikleri ve kavramsal olarak neyi yada neleri belirtebilecekleriydi. C# dilinde soyutlamaları anlamaya çalıştığım ilk zamanlarda, aynı satırları 4 hatta 5 kez okuduğumu ve uzun süreler ne olup bittiğini anlamak için çabaladığımı hatırlıyorum.

Bu sıkıntıyı daha önce yaşadığımdan, şimdi java dilinide soyutlamaları incelerken de benzer durumlar ile karşılaşabileceğim korkusu ile bu haftaki soyutlama çalışmalarımı daha bir titiz yapma zorunluluğunu hissettim. Neyseki C# dilinden pek bir farkı yokmuş.

En önemli konu, soyutlamanın tam olarak sınıflara ve içerisinde yer alan metodlara uygulandığını bilmek. Peki o halde soyut sınıf nedemek? Soyut sınıfları, somut olmayan sınıflar olarak tanımlıyarak bu kavramı açıklamaktan kurtulabilirim. Keza, somut olmayan sınıflarda soyutturlar. Bu aslında, Japonların yada Çinlilerin (hangi tarafın bu fikrin orjinaline sahip olduğunu bilmiyorum), Ying Yang felsefesinin bir sonucudur. Ancak bu tanımlama her nekadar geçerli bir tanımsada, fazla bir şey ifade etmemektedir.

Asıl olan, soyut sınıfların, kalıtımın bir parçası olarak normal bir sınıfın sahip olabileceği üyeler dışında, soyut metodlara sahip oluğu ve bu soyutlamadan türeyen sınıflardada, soyutlanmış metod bildirimlerini mutlaka ve mutlaka uygulamak zorunda bıraktığı yapılardır. Bu açıdan bakıldığında, soyut sınıfların, kalıtımda birleştirici bir rol üstlenmenin yanı sıra, türeyen sınıfların zorunlu olarak izlemeleri gereken bir rehber oluşturduğunuda söyleyebiliriz.

Soyutlamanın bu uzun ama çok şey ifade eden tanımı yinede bu konuyu anlamak için yeterli değildir. Nitekim, soyutlamda

dikkat edilmesi gereken ve göze çarpan pek çok nokta vardır. Ama yinede, şu an için, soyutlamayı gösteren basit bir örnek ile işe girişmenin daha faydalı olacağı kanısındayım. Her şeyden önce, soyut sınıfların ve metodların nasıl tanımlandığını bilmemiz gerekiyor. Kaldıki soyutlamanın önemli yapıtaşlarını oluşturan kurallar bu temeller üzerine kurulu olucaktır. Bu amaçla, aşağıdaki gibi basit bir örnek geliştirdim.

```
abstract class SoyutTemel
{
    public static void Yaz()
    {
        System.out.println("Ben soyut temel sinifim");
    }

    abstract public void Kimlik();

    SoyutTemel()
    {

    }

    int deger=40;
}

class Tureyen1 extends SoyutTemel
{
    public void Kimlik()
    {
        System.out.println("Ben SoyutTemel siniftan tureyen
Tureyen1");
    }
}

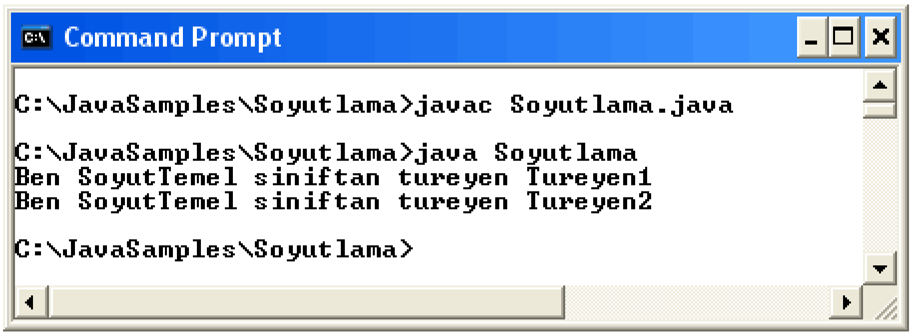
class Tureyen2 extends SoyutTemel
{
    public void Kimlik()
    {
        System.out.println("Ben SoyutTemel siniftan tureyen
Tureyen2");
```

```
   }
}

public class Soyutlama
{
   public static void main(String[] args)
   {
      Tureyen1 t1=new Tureyen1();
      Tureyen2 t2=new Tureyen2();

      t1.Kimlik();
      t2.Kimlik();
   }
}
```

Örneği derleyip çalıştırdığımda aşağıdaki ekran görüntüsünü elde ettim.



Bu uygulamada ilk dikkati çeken nokta soyutlamanın uygulandığı sınıfların tanımlanma şeklidir. Bir sınıf soyut olucaksa, abstract anahtar sözüğü ile tanımlanır.

```
abstract class SoyutTemel
```

Bu anahtar sözcük ile SoyutTemel sınıfının soyutlamayı uygulayacağını belirtmiş oluruz. Peki bu uygulamanın getirdiği etkiler nelerdir? İlk olarak şunu söyleyebilirim. Soyut bir sınıftan herhangibir nesne örneği türetilemez. Bunun sebebi, soyut sınıfların birleştirici özelliklerinin yanısıra, abstract metod tanımlamarını içermesidir.

```
abstract public void Kimlik();
```

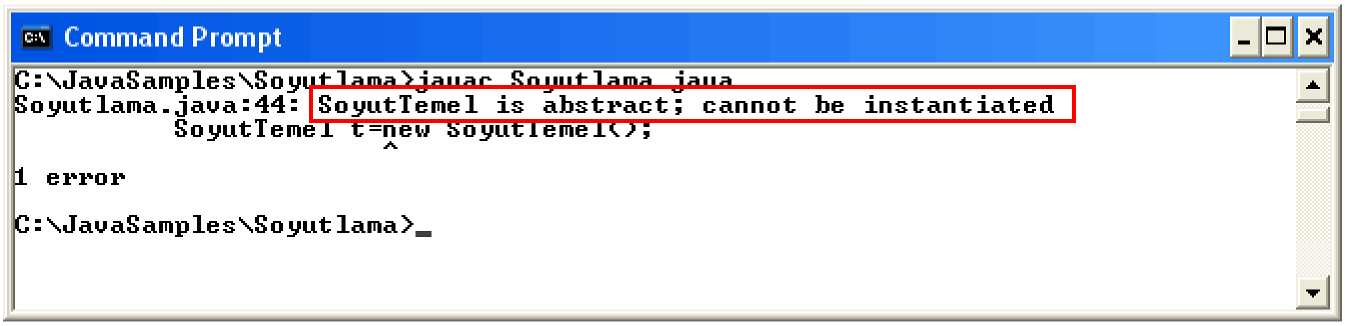
Burada görüldüğü gibi abstract metodlar, sadece bir metod bildiriminden ibarettir. Öyleki bu metod bildirimi, bu soyut sınıftan türeyen sınıflarda mutlaka ama mutlaka override edilmelidir. Dolayısıyla, eğer bir soyut sınıf nesne örneği oluşturulabilseydi, bu metod bildirimlerine erişilmesi son derece anlmasız olurdu. İşte bu nedenle, soyut sınıfların örneklendirilmesi yasaklanmıştır.

Bununla birlikte bu yasaklama, sanıldığı gibi, soyut sınıfların herhangibir yapılandırıcı içeremeyeceğini göstermez. Nitekim yukarıdaki örneğimizde, abstract sınıfımız için bir adet varsayılan yapıcı metod yazılmıştır. O halde, java dili abstract bir sınıfın nesne örneğini, new operatörü kullanıldığı yerde denetleyecektir. Örneğin, uygulamamızın Main metodu içerisinde SoyutTemel abstract sınıfından bir nesne örneği yaratmaya çalışalım. Bu durumda java derleyicisi derleme zamanında bizi bir hata ile uyaracaktır.

```
public static void main(String[] args)
{
    Tureyen1 t1=new Tureyen1();
    Tureyen2 t2=new Tureyen2();

    t1.Kimlik();
    t2.Kimlik();

    SoyutTemel t=new SoyutTemel();
}
```



Diğer yandan, soyut sınıflara ait nesne örnekleri tanımlayamasakta, soyut sınıflara ait nesneler tanımlayarak (yani sadece nesne tanımlama, oluşturmak değil) bu nesnelere, türemiş sınıf nesnelerinin referanslarını aktarabiliriz. Bu bize,

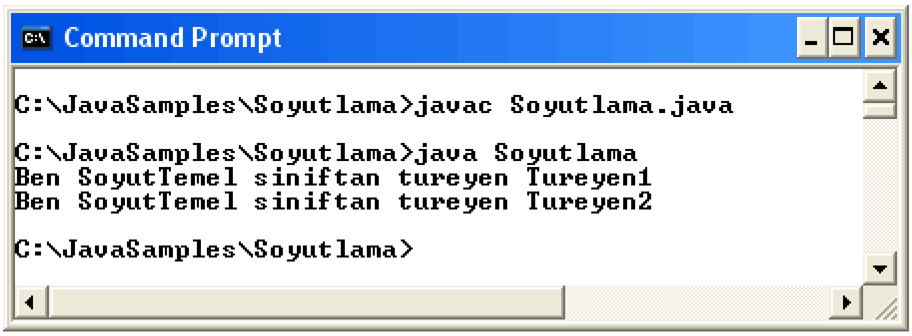
kalıtımın en önemli taşlarından olan çok biçimliliği uygulama fırsatını verir ve dostumuz bugi'yi hatırlamamızı sağlar. Nitekim, soyut sınıflar, kalıtımın bir parçası olarak, çok biçimliliğide bir şekilde desteklemelidirler. Bu kalıtımın doğası gereği ortaya çıkan bir sonuçtur. Bu anlamda yukarıdaki örneğe çok biçimliliği aşağıdaki gibi uyguladım.

```
public class Soyutlama
{
    public static void Yazalim(SoyutTemel[] t)
    {
        for (int i=0;i<t.length;i++)
        {
            t[i].Kimlik();
        }
    }
    public static void main(String[] args)
    {
        Tureyen1 t1=new Tureyen1();
        Tureyen2 t2=new Tureyen2();

        SoyutTemel[] dizi=new SoyutTemel[2];
        dizi[0]=t1;
        dizi[1]=t2;

        Yazalim(dizi);
    }
}
```

Uygulamayı bu haliyle derleyip çalıştırdığımda aşağıdaki ekran görüntüsünü elde ettim.



Burada çok biçimliliği uygularken, SoyutTemel sınıfından bir nesne dizisi kullandım. Ancak buradaki new operatörü, bir nesneye ait yapılandırıcıları çağıran new operatörü ile aynı değildir. Tek yaptığımız, SoyutTemel sınıfından nesneler taşıyacak iki elemanlı bir dizi tanımlamaktır. Keza, bu dizinin elemanları, abstract sınıfımızdan türeyen sınıf nesne örnekleridir. Çok biçimliliği, Yazalim metodunda uyguluyoruz. Burada SoyutTemel abstract sınıfından nesneler, kendisine atanan türeyen sınıflara ait Kimlik metodlarını çağırıyor. İşte böylece abstract sınıflarada çok biçimliliği uygulatabildiğimizi görmüş oluyoruz.

Abstract sınıf tanımlamalarında dikkat çeken bir diğer unsurda, bu sınıfın abstract metodlar dışındada normal sınıfların sahip olduğu üyelere sahip olabilmesidir. Normal metodlar veya alanlar. Normal bir metodu türeyen sınıflara ait nesneler üzerinden kalıtımın bir gereği olarak çağırabiliriz. Ancak bu metodları, abstract sınıfından bir nesne örneği üzerinden çağıramıyacağımızda kesindir. Çünkü, abstract sınıflardan bir nesne örneği oluşturamayız. Böyle bir durumda tanımlandığı sınıfın nesne örneğine ihtiyaç duymayan static metodları kullanabiliriz. Bu aslında, herhangibir örneğinin kesinlilikle türetilmesini istemediğimiz nesnelere ait bir takım metodları uygulatmakta kullanılabileceğimiz bir tekniktir.

Gelelim abstract sınıflardaki yapıcı metodların işleyişine. Bir abstract sınıfa ait nesne örneğinin kesinlikle oluşturulamıyacağını biliyoruz. Ancak bir abstract sınıfın yapıcılar içerebildiğinide bilmekteyiz. Bununla birlike bitmek tükenmek bilmeyen bilgilerimiz yanında, abstract sınıftan türeyen bir sınıfın nesne örneğinin, kalıtımın doğası gereği ilk olarak abstract sınıf içerisindeki yapıcı metodu işleyeceğinide biliyoruz. Buraya kadar aslında benim içinde enterasan gelebilecek bir olgu yok. Ancak aşağıdaki gibi bir kod ne gibi sonuçlar doğurabilir?

```
SoyutTemel()
{
   Kimlik();
}
```

Burada abstract sınıfın içerisinden, abstract olarak tanımlanmış Kimlik metodu çağırılmıştır. Bu metodun çağırılacağı kesindir. Çünkü, abstract sınıftan türetilen bir sınıfın nesne örneği oluşturulurken önce, abstract sınıftaki yapıcı ve bu yapıcı içindende, türeyen sınıftaki override edilmiş abstract metod çağırılacaktır. Ancak işin içine, yapıcılar içerisinde ilk değer atamaları girdiğinde durum biraz garipleşir. Bunu açıklayabilmem için aşağıdaki gibi bir örnek geliştirmem gerekti.

```
abstract class SoyutTemel
{
   public static void Yaz()
   {
      System.out.println("Ben soyut temel sinifim");
   }

   abstract public void Kimlik();

   SoyutTemel()
   {
      System.out.println("Soyut metoddan once");
      Kimlik();
      System.out.println("Soyut metoddan sonra");
   }
   int deger=40;
}

class Tureyen1 extends SoyutTemel
{
   int No;
   public void Kimlik()
   {
      System.out.println("Ben SoyutTemel siniftan tureyen
Tureyen1 Kimlik No:"+No);
   }
   Tureyen1()
   {
      System.out.println("TUREYEN 1 YAPICISI");
      No=154528;
```

```
    }
}

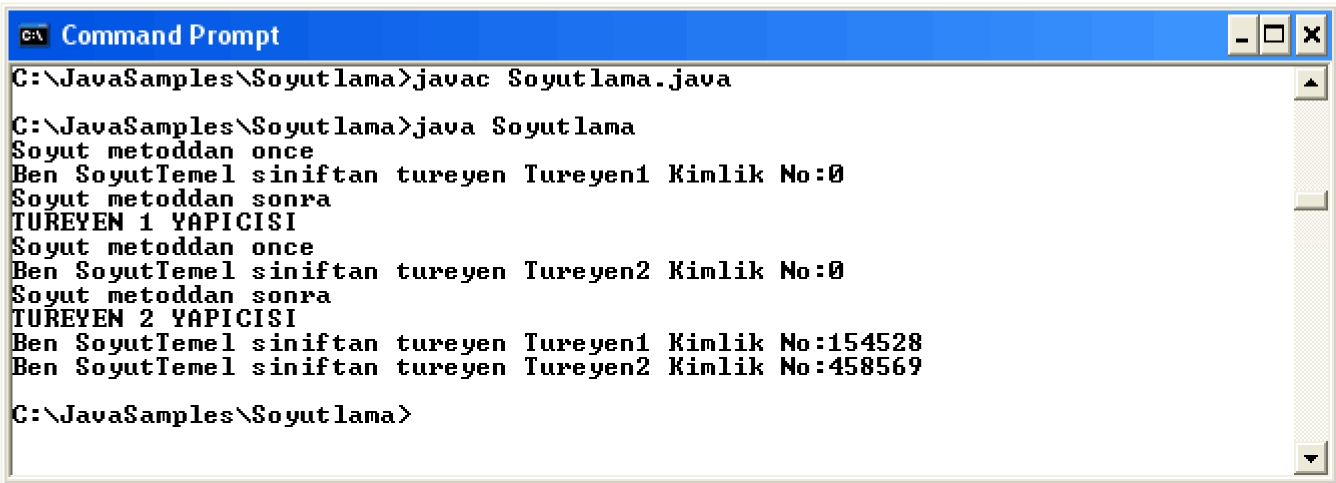
class Tureyen2 extends SoyutTemel
{
    int No;
    public void Kimlik()
    {
        System.out.println("Ben SoyutTemel siniftan tureyen
Tureyen2 Kimlik No:"+No);
    }
    Tureyen2()
    {
        System.out.println("TUREYEN 2 YAPICISI");
        No=458569;
    }
}

public class Soyutlama
{
    public static void Yazalim(SoyutTemel[] t)
    {
        for (int i=0;i<t.length;i++)
        {
            t[i].Kimlik();
        }
    }
    public static void main(String[] args)
    {
        Tureyen1 t1=new Tureyen1();
        Tureyen2 t2=new Tureyen2();

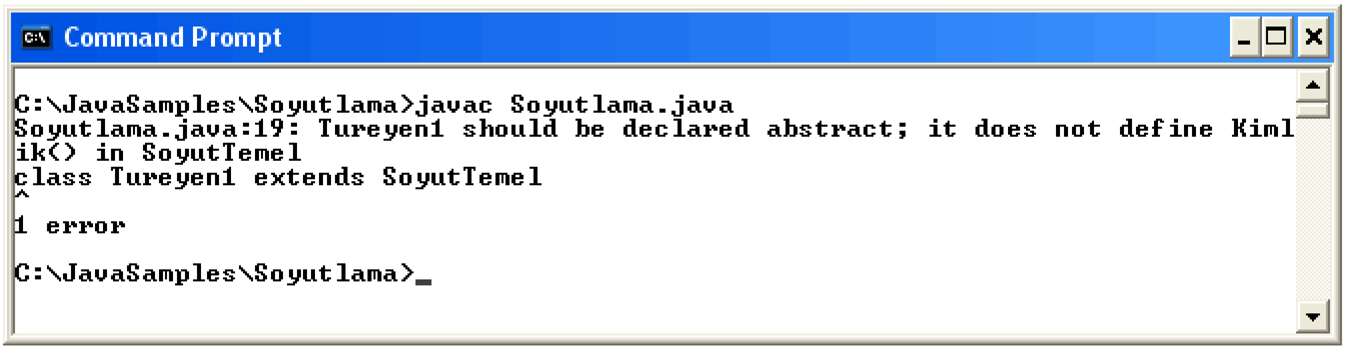
        SoyutTemel[] dizi=new SoyutTemel[2];
        dizi[0]=t1;
        dizi[1]=t2;

        Yazalim(dizi);
    }
}
```

Öncelikle uygulamayı inceleyelim. Tureyen sınıflardan bir nesne oluşturulduğunda, kalıtımın bir gereği olarak, ilk önce SoyutTemel sınıfının yapıcısı çağırılacaktır. Burada ise, türeyen sınıflarda override edilen Kimlik metodu çağırılmıştır. Kimlik metodu, override edildiği sınıfın (Tureyen1,Tureyen2) yapıcısındaki No değerini ekrana yazdırmalıdır. Ancak durum böyle olmaz. Çünkü henüz, SoyutTemel abstract sınıfının yapıcısı sonlandırılmamıştır. Bu nedenle, No alanın değeri, java tarafından integer değişkenlerin alacağı varsayılan değeridir, yani 0'dır. Dolayısıyla override edilen metod içinden, No alanının ilgili türeyen sınıf yapıcısında belirlenen değeri elde edilememiştir. Ancak abstract sınıfın yapıcısı işlevini bitirdikten sonra, türeyen sınıftaki yapıcı çağırılacak ve No alanının değeri yapıcılarda belirlenen halini alıcaktır. Bu nedenle uygulamanın ekran çıktısı aşağıdaki biri olur.



Soyut sınıfların, kalıtımda, türeyen sınıflar için birleştirici bir rol üstlenerek, onların uygulaması gereken metodları belirten bir rehber olduğunu söyleyebiliriz. Burada türeyen sınıflara yönelik bir zorlamada söz konusudur. Nitekim, abstract sınıfların tanımlanması ile oluşturulan abstract metodlar, mutlaka ve mutlaka türeyen sınıflarda override edilmelidir. Söz gelimi yukarıdaki örneğimizde, Tureyen1 sınıfında Kimlik metodunu override etmediğimizi düşünelim. Bu durumda aşağıdaki gibi bir derleme zamanı hatası alırız.



Görüldüğü gibi derleyici, Kimlik metodunun abstract olmasına rağmen, türeyen sınıf içerisinde tanımlanmadığını belirtiyor. Oysaki normal bir sınıfın kalıtımda temel sınıf olarak rol oynamasında, türeyen sınıfların, temel sınıftaki metodları override etmek gibi bir zorunluluğu yoktur. İşte buda abstract sınıfları normal sınıflardan ayıran diğer bir olgudur.

Soyutlama ile ilgili temelleri aştığımı düşündüğüm bir sırada kafama oldukça güzel sorular geldi. Nerde geldiğini sormayın nitekim bunları düşünürken, **W**inston **C**hurchill ile görüşmedeydim. Doğruyu söylemek gerekirse bu adam her zaman dimamı açıyor.

Acaba bir abstract sınıftan başka abstract sınıflar türetebilir miydim? Türetirsem, türeyen abstract sınıflarda, abstract olmanın bir gereği olarak, temel abstract sınıftaki abstract metodları bildirmem gerekir miydi? Peki ya böyle bir hiyerarşide, en altta yer alan abstract sınıftan normal bir sınıf türetirsem, bu sınıf içinde, hiyerarşideki abstract metodların akibeti ne olurdu? Önce işe bir abstract sınıftan başka bir abstract sınıf oluşturmakla başladım.

```
abstract class TemelSoyut
{
    abstract public void Metod1();
    abstract public int Metod2(int a,int b);
    abstract public double Sec();

    public static void Yaz()
    {
        System.out.println("BEN ABSTRACT TEMEL SINIFIM");
    }
```

```
}

abstract class TureyenSoyut1 extends TemelSoyut
{

}
public class Soyutlama2
{
    public static void main(String[] args)
    {

    }
}
```

Öncelikle programın başarılı bir şekilde derlendiğini söyleyebilirim. İlk aşamada iki sorumun cevabını almıştım. Soyut sınıflardan yeni soyut sınıflar türetilebiliyordu. Bununla birlikte, türeyen soyut sınıf içerisinde, temel soyut sınıfta tanımlanmış abstract metodların override edilme zorunluluğu gibi bir durum söz konusu olmamıştı. Gelelim asıl önemli noktaya. Yani türeyen abstract sınıftan bir başka sınıf türetmeye. Örneği biraz daha geliştirdim. Türeyen abstract sınıf içerisinede başka abstract metod bildirimleri ekledim.

```
abstract class TemelSoyut
{
    abstract public void Metod1();
    abstract public int Metod2(int a,int b);
    abstract public double Sec();

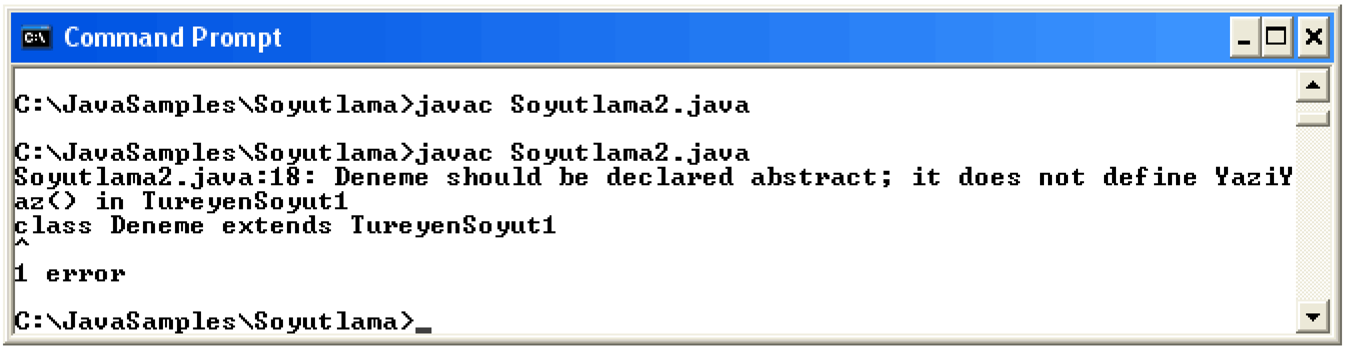
    public static void Yaz()
    {
        System.out.println("BEN ABSTRACT TEMEL SINIFIM");
    }
}

abstract class TureyenSoyut1 extends TemelSoyut
{
    abstract public void YaziYaz();
}
```

```
class Deneme extends TureyenSoyut1
{

}
```

Bu haliyele uygulamayı derledim ve aşağıdaki gibi bir hata mesajı aldım.



Oldukça şaşırtıcı bir durumla karşılaşmıştım. Derleyici şu an için sadece TureyenSoyut1' deki YaziYaz abstract metodunu override etmem gerektiğini söylüyordu. Oysaki, TureyenSoyut1 abstract sınıfda başka bir abstract sınıftan türemişti. Benim umduğum, TemelSoyut sınıfında yer alan ve override edilmeyen abstract metodlar içinde hatalar alabilmekti. Bana düşen öncelikle YaziYaz metodunu override etmekti.

```
class Deneme extends TureyenSoyut1
{
   public void YaziYaz()
   {
      System.out.println("YAZI YAZIYORUM");
   }
}
```

Şimdi uygulamayı derlediğimde, ilk defa gördüğüm bir derleme hatası için bu kadar çok sevinmiştim. Bu kez derleyici, TureyenSoyut1 abstract sınıfının türediği TemelSoyut sınıfındaki Sec metodunun override edilmediğini belirtiyordu. Dünyalar benim oldu desem yeridir.

```
C:\JavaSamples\Soyutlama>javac Soyutlama2.java
Soyutlama2.java:18: Deneme should be declared abstract; it does not define Sec()
 in TemelSoyut
class Deneme extends TureyenSoyut1
^
1 error

C:\JavaSamples\Soyutlama>
```

Bu hatalar sinsilesinin tek handikapı böyle tek tek ekrana gelmesiydi. Yani aslında hata mesajından, override edilmesi gereken tüm metodlar için bir uyarı alamıyordum. Yinede, türetilen abstract sınıflardan türeyen sınıfların abstract sınıf hiyerarşisindeki tüm metodları override etmesi gerektiği sonucuna varabilmiştim.

Soyut sınıflar ile ilgili olarak dikkat edilmesi gereken pek çok nokta var. Bu noktaları iyice kavrayıncaya kadar, aşağıdaki tabloda görüldüğü gibi not etmeyi uygun görüyorum. Eğitimde ezberciliğe her zaman karşıyımdır. Üniversite yıllarındayken, kalın not defteleri ile birlikte öğrendiklerimi yanımda taşırdım. Taki öğrendiklerim üzerinde yorumlar ve felsefik söylemlerde bulununcaya dek. O günler geldiğinde not defterlerini rafa kaldırırdım. Şimdi ise, yanımda not defteri yerine Laptop taşıyorum. Ama yinede ilk öğrendiğim ve üzerlerinde henüz felsefik söylemler yapamadığım kavramları dosyalarda saklıyor ve yeri geldikçe bakıp hatırlıyorum. İşte aşağıdaki tabloyuda bu yaklaşım ile hazırladım.

Soyutlamada Dikkate Değer Noktalar	
1	Soyut sınıflardan nesne örnekleri oluşturulamaz.
2	Bir sınıfın soyut olabilmesi için en azından bir tane soyut metod bildirimine sahip olması gerekir.
3	Soyut sınıflardan türetilen sınıflar, soyut metod bildirimlerini mutlaka override etmelidirler.
4	Soyut sınıflardan türeyen nesne örneklerini, soyut sınıf nesnelerine referans olarak aktararak çok biçimliliğin uygulanmasını sağlayabiliriz.
5	Soyut sınıflardan başka soyut sınıflarda türetilebilir.

6	Soyut metodları private olarak tanımlayamayız.
7	Soyut sınıflara ait yapıcılar tanımlayabiliriz. Ancak bu yapıcıları soyut tanımlayamayız.
8	Static bir soyut metod bildirimi yapamayız.
9	Soyut sınıflar içerisinde soyut metod bildirimleri dışında, normal metodlar ve alanlar tanımlayabiliriz.

Bu hafta boyunca soyutlama ile ilgili bu kavramları inceledikten sonra, nesne yönelimli programlama dillerinde soyutlama ile çok sık karıştırılan bir kavramı incelemeye karar verdim. Sanırım bir sonraki kahve molamda, Interface (arayüz) kavramını inceleyeceğim.

Bölüm 11: Arayüzler (Interfaces)

Geçen hafta boyunca, nesne yönelimli programlama dillerinde, aralarındaki farklılıklar dışında kavramsal olarak birbirlerinden ayrılmakta zorlanan kavramlardan birisi olan soyutlamanın Java dilindeki yerini araştırmıştım. Soyutlama ile ilgili olarak incelemelerimi tamamladıktan sonra, sırada arayüzlerin incelenmesine vardı. C# programlama dilinden aşina olduğum arayüzler, java dilindede yer almaktaydı. Her zaman olduğu gibi karşımda bu iki kavram için sorulan o meşhur sorular topluluğu bulunuyordu. Neden arayüz, neden soyutlama, hangisini kullanmalıyım, kim daha avantajlı vb...?

Nesne yönelimli dilleri geliştirenlerin bile oldukça karmaşık bir biçimde cevaplayabilecekleri yada açıklayabilecekleri bu ayrımı anlamak için ne kadar çaba göstersemde, sonuçlar hiç bir zaman iç açıcı olmamıştı. Ancak en azından, arayüzlerin kullanılması ile soyutlama arasındaki temel farklılıkları iyi bilmem gerektiği kanısındaydım. İşe öncelikle arayüzlerin ne olduğunu tanımlamakla başlamakta fayda vardı.

Arayüzlerin en belirgin özelliği, soyut metod tanımlamalarında olduğu gibi metod bildirimleri içermesiydi. Dolayısıyla arayüzlerin, diğer sınıflar için yol gösterici olacak şekilde tanımalanan bir yapı olduğunu söyleyebilirim. Bu tanım

itibariyle, soyut sınıflar ile arasında çok büyük bir benzerlik var. Her iki teknikte, kendilerini uygulayan sınıflara rehberlik etmek üzere ortak bildirimlere izin veriyor. Soyutlamada bu, soyut sınıflar içerisindeki soyut metodlar ve kalıtımın bir arada ele alınması ile gerçekleştiriliyor. Arayüzlerde ise durum daha farklı. Nitekim arayüzler, soyut sınıflarda olduğu gibi iş yapan metodlar içermiyor. Onun yerine sadece metod bildirimlerini ve alan tanımlamalarını içeren ve kalıtım yerine implementasyonun bir arada ele alındığı bir teknik söz konusu.

İşte bu noktada daha olayın başında, arayüzler ile soyutlama arasındaki en temel farkı görmüş oldum. Soyutlamada soyut metod bildirimleri haricinde iş yapan metodların var olması söz konusu iken, arayüzlerde sadece metod bildirimleri yer almakta. Şu anda, bir arayüzün nasıl oluşturulduğunu ve bir sınıf tarafından nasıl kullanıldığını açıkça görme ihtiyacı hissediyorum. İşte bu isteğin karşılığında, her zaman olduğu gibi anlamsız herhangibir iş yapmayan ama kavramı açıklayan bir kod geliştirmem gerektiği kanısına varıyorum. Olayı sade ve basit olarak düşünmek ve kavramlar üzerinde yoğunlaşmak şu an ihtiyacım tek şey.

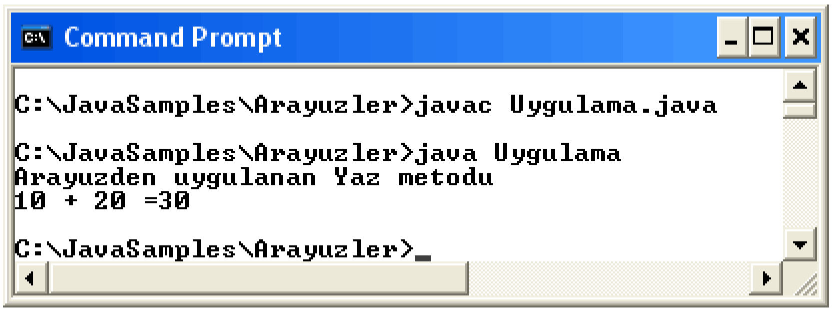
```
interface IArayuz
{
   void Yaz();
   int Topla(int a,int b);
}

class Mat implements IArayuz
{
   public void Yaz()
   {
      System.out.println("Arayuzden uygulanan Yaz metodu");
   }
   public int Topla(int deger1,int deger2)
   {
      return deger1+deger2;
   }
}

public class Uygulama
```

```
{
   public static void main(String[] args)
   {
      Mat m=new Mat();
      m.Yaz();
      int topla=m.Topla(10,20);
      System.out.println("10 + 20 ="+topla);
   }
}
```

Uygulamayı derleyip çalıştırdığımda aşağıdaki sonucu elde ettim.



Sonuçtan daha çok benim için önemli olan, arayüz tanımlamasının yapılış şekli ve bu arayüzün bir sınıfa uygulanışıydı. Bir arayüz interface anahtar kelimesi ile tanımlanan ve sisteme bir class dosyası olarak kaydedilen bir yapıdır.

```
interface IArayuz
```

Diğer yandan bu arayüzü bir sınıfa uygulamak istediğimde, implements anahtar sözcüğünü kullanmam gerekiyor.

```
class Mat implements IArayuz
```

Implements anahtar sözcüğü ile bu sınıfın belirtilen arayüzü uygulayacağını ve bu nedenlede belirtilen arayüz içindeki tüm metodları uygulaması gerektiğini söylemiş oluyoruz. C# dilinden kalma bir alışkanlık olarak tanımladığım arayüzün bir Interface (arayüz) olduğunu daha kolay anlayabilmek amacıyla, I harfini arayüz adının başına ekledim. Tanımladığım Mat sınıfı içinde arayüzdeki tüm metodları override ettim. Bu bir zorunluluk.

Eğer, arayüz içinde tanımlı metodlardan override etmediğim olursa, söz gelimi Yaz metodunu override etmessem aşağıdaki gibi bir derleme zamanı hatası alırım.

```
Command Prompt
C:\JavaSamples\Arayuzler>javac Uygulama.java
Uygulama.java:7: Mat should be declared abstract; it does not define Yaz() in Ma
t
class Mat implements IArayuz
^
1 error

C:\JavaSamples\Arayuzler>
```

Arayüzlerde tanımlanan metod bildirimleri ile, soyut sınıflarda tanımlanan soyut metodlar arasındada belirgin farklılıklar mevcut. Herşeyden önce arayüzler içindeki metod bildirimlerinde abstract anahtar sözcüğünü kullanmıyoruz. Bununla birlikte dikkatimi çeken bir diğer önemli noktada, arayüz metod bildirimlerinin herhangibir erişim belirleyicisini kabul etmediği. Bu metodlar varsayılan olarak public kabul ediliyorlar ve public dışındaki bir erişim belirleyicisi ile tanımlanamıyorlar. Bu aslında arayüzlerin içerdiği metod bildirimlerinin, diğer sınıflara uygulandıklarında override edilebilmelerinin bir gerekliliği olarak düşünülebilir. Uygulamamda kullandığım arayüzdeki metodlarda başka türden erişim belirleyicileri kullanmaya çalıştığımda derleme zamanı hataları ile karşılaştım.

```
interface IArayuz
{
    private void Yaz();
    protected int Topla(int a,int b);
}
```

```
Command Prompt
C:\JavaSamples\Arayuzler>javac Uygulama.java
Uygulama.java:3: modifier private not allowed here
        private void Yaz();
                     ^
Uygulama.java:4: modifier protected not allowed here
        protected int Topla(int a,int b);
                      ^
2 errors
```

Gördüm ki arayüz içindeki metodlar public olmak zorundalar. Zaten bu sebepten dolayıda varsayılan erişim belirleyicileri, arayüzlerdeki metodlar için public olarak belirlenmiş.

Arayüzlerin kullanılmalarının en önemli nedenlerinden birisde sınıflar üzerinde çoklu kalıtıma izin vermeleri. Normalde, C# dilinde sınıflar arasında çoklu kalıtıma izin verilmediğini ve bu işin arayüzler yardımıyla gerçekleştirildiğini biliyordum. Java dilinde de durumun böyle oluşu beni çok fazla şaşırtmadı açıkçası. Kahvemin bir sonraki yudumu çoklu kalıtım ile ilgili bir örneği geliştirmem gerektiği konusunda beyin hücrelerimi uyarıyordu. Ah keşke birazda daha işe yarar örnekler oluşturabilmeme yardımcı olsa şu kafein mereti. Neyse, önemli olan kavramları iyi anlayabilmek ve nasıl uygulandıklarını teorik olarak en basit şekilde ifade edebilmek değilmi? O halde ne duruyorum.

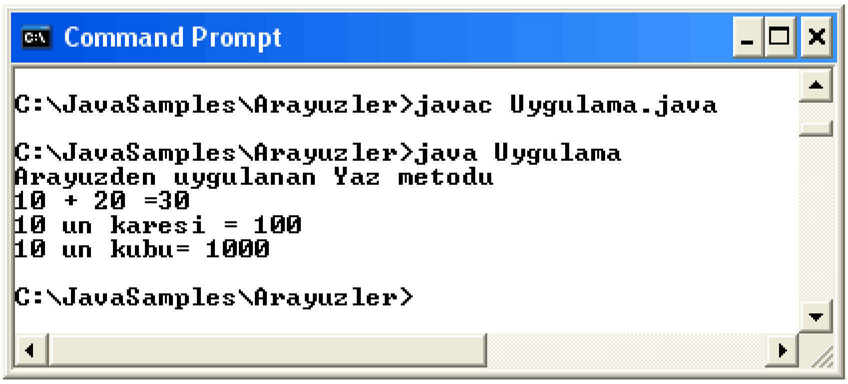
```
interface IArayuz
{
    void Yaz();
    int Topla(int a,int b);
}

interface IArayuz2
{
    int Karesi(int a);
    int Kup(int a);
}

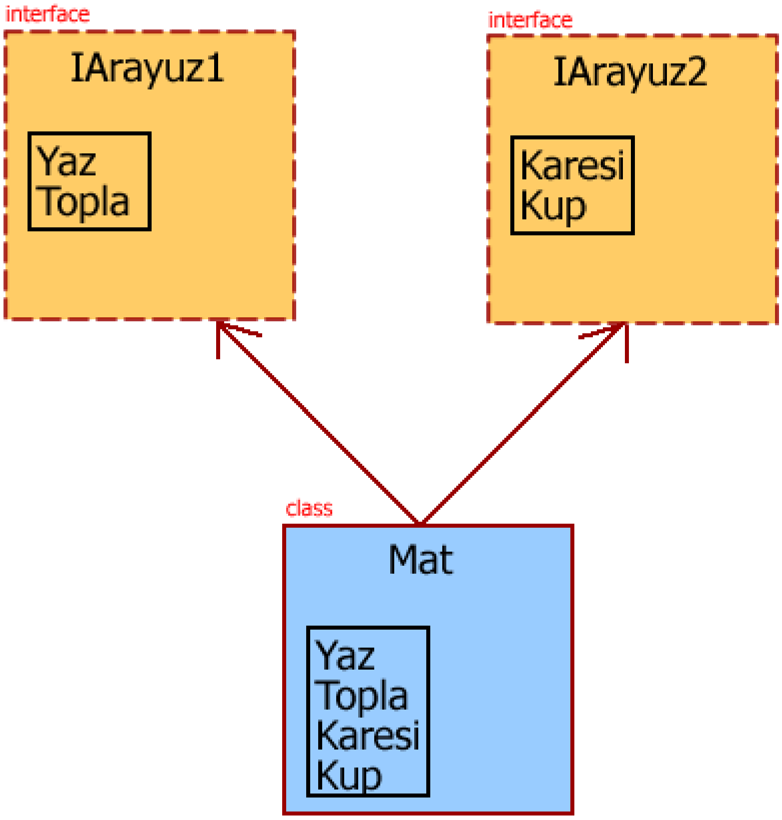
class Mat implements IArayuz,IArayuz2
{
    public void Yaz()
    {
        System.out.println("Arayuzden uygulanan Yaz metodu");
    }
    public int Topla(int deger1,int deger2)
    {
        return deger1+deger2;
    }
    public int Karesi(int deger)
    {
```

```
        return deger*deger;
    }
    public int Kup(int deger)
    {
        return deger*deger*deger;
    }
}

public class Uygulama
{
    public static void main(String[] args)
    {
        Mat m=new Mat();
        m.Yaz();
        int topla=m.Topla(10,20);
        System.out.println("10 + 20 ="+topla);
        System.out.println("10 un karesi = "+m.Karesi(10));
        System.out.println("10 un kubu= "+m.Kup(10));
    }
}
```



Burada yaptığım, IArayuz1 ve IArayuz2 interface�lerini, Mat isimli sınıfa uygulamak. Bu şekilde, çoklu kalıtımı sağlamış oldum. Yani bir sınıf birden fazla arayüz üyesini override ederek bu hakkı kazanmış oldu. Yukarıdaki uygulamanın çalışma şeklini daha iyi anlamak amacıyla da aşağıdaki gibi bir şekil geliştirdim.



Arayüzler ile ilgili bir diğer önemli konu ise, farklı arayüzlerin aynı isimli metodlar barındırmaları sırasında ortaya çıkabilecek sorunlar. Bunu daha iyi anlayabilmek için, soruna neden olucak bir örnek geliştirmem gerekiyor. Öyleki, aynı isimli metod bildirimlerini içeren iki arayüzü bir sınıfa uygulamaya çalıştığımda, bir hata mesajı almalıyım. Gelmek istediğim nokta aslında overloading kavramının temel niteliklerinden birisi. Ama önce örneği yazıp üzerinde düşünmek gerektiği kanısındayım.

```
interface IArayuzYeni
{
    void Metod1(String metin);
}

interface IArayuzDiger
{
    int Metod1(String a);
    void Metod1(String metin, int deger);
}
class UygulayanSinif implements IArayuzYeni, IArayuzDiger
{
    public void Metod1(String yazi)
```

```
    {

    }

    public void Metod1(String yazi, int sayi)
    {

    }

    public int Metod1(String metin)
    {
        return 3;
    }
}
public class Uygulama2
{

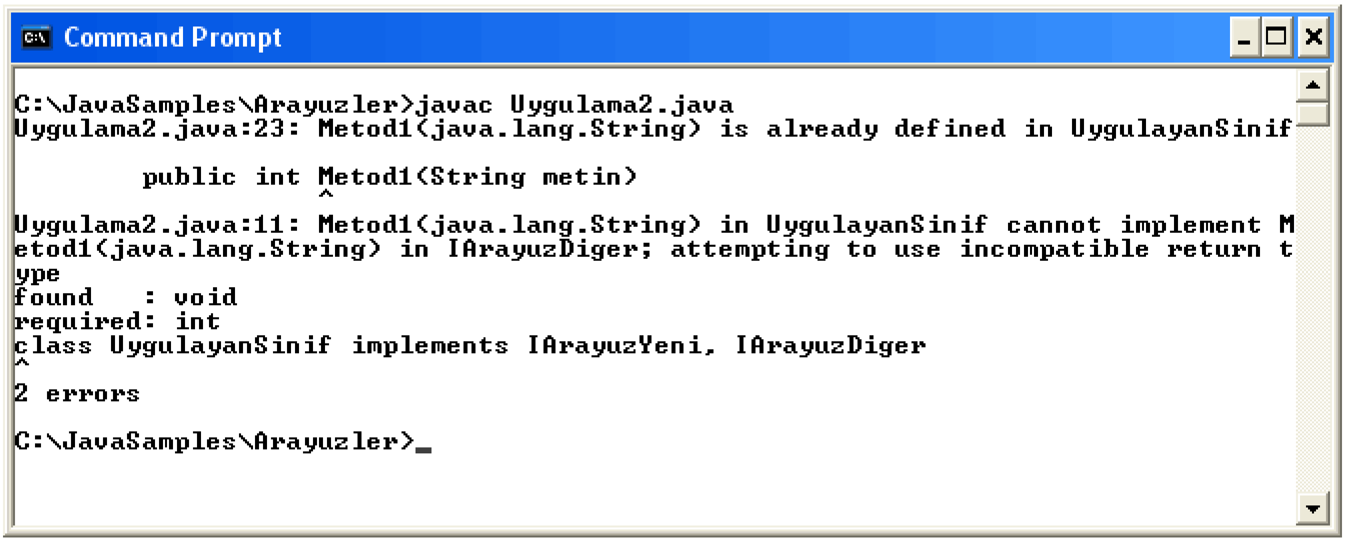
}
```

Burada iki arayüz içerisinde Metod1 isminde 3 farklı metod bildirimi yapılmıştır. İlk aşamada, bu arayüzlerin ikisinide birlikte uygulayan sınıfımız için bir sorun çıkmayacağı düşünülebilir. Ancak metodların bu şekilde aşırı yüklenmelerinde metod imzalarının önemli olduğunu hatırlamakta fayda var. Öyleki, burada her iki arayüzde de ayrı ayrı tanımlanmış

```
void Metod1(String metin);

int Metod1(String a);
```

metod bildirimleri farklı arayüzlerde olasalar dahi, aynı sınıfa uygulandıklarından aşırı yüklenmiş metodların sahip olacakları karakteristikleri taşımaları gerkiyor. Nitekim burada bu metodların aşırı yüklenmesini engelleyen bir olgu var ki buda metodların imzaları. Metod isimleri ve parametreler tamamen aynı, fakat metodların geri dönüş değerinin farklı olmasının metodun imzasında etkin rol oynamaması, aşağıdaki derleme zamanı hatalarının oluşmasına neden olmakta.



Gelelim arayüzler ile ilgili diğer bir noktaya. Arayüzleri oluşturduğumuzda, bunların sisteme birer class dosyası olarak kaydedildiğini görürüz. Dolayısıyla arayüzlerin birer sınıf olduğunu düşünebiliriz. Bununla birlikte, sınıflarda olduğu gibi arayüzleri de birbirlerinden türetebilir ve böylece arayüzler arasında kalıtımın gerçekleşmesini sağlayabiliriz. Örneğin,

```
interface ITemel
{
   void Temel();
}

interface ITureyen1 extends ITemel
{
   void Tureyen1();
}

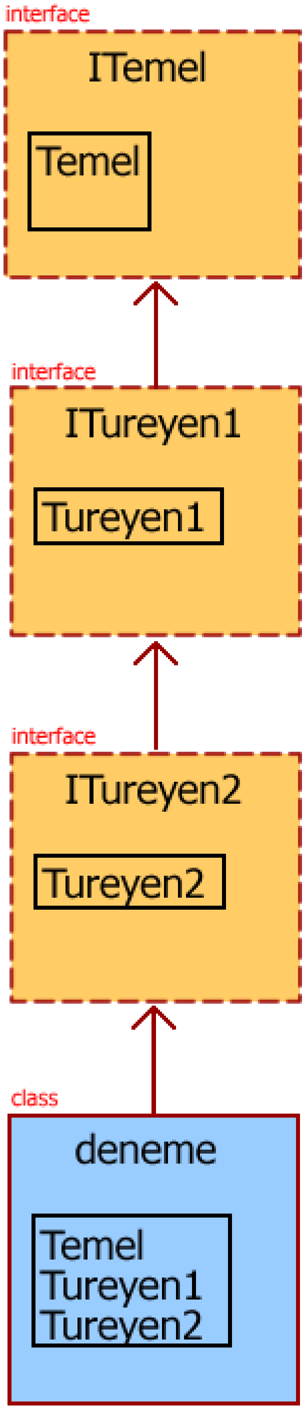
interface ITureyen2 extends ITureyen1
{
   void Tureyen2();
}

class deneme implements ITureyen2
{
   public void Temel()
   {
      System.out.println("Temel arayuzun metodunu
uyguladim...");
```

```
   }
   public void Tureyen1()
   {
      System.out.println("Tureyen1 arayuzunun metodunu
uyguladim...");
   }
   public void Tureyen2()
   {
      System.out.println("Tureyen2 arayuzunun metodunu
uyguladim...");
   }
}

public class Uygulama3
{
   public static void main(String[] args)
   {
      deneme d=new deneme();
      d.Temel();
      d.Tureyen1();
      d.Tureyen2();
   }
}
```

Bu örnekte, deneme isimli sınıf sadece ITureyen2 arayüzünü uyguluyor. Ancak bu arayüz kalıtım yolu ile, IArayuz1 interface�inden, IArayuz1 ise, ITemel arayüzünden türetilmiş durumda. Bu nedenle, deneme sınıfının tüm bu arayüzlerdeki metodları override etmesi gerekiyor. Olayı daha fazla dramatize etmeden şematizme etmenin daha faydalı olacağını düşünerek aşağıdaki umlvari grafiği hazırladım.



Kalıtımın arayüzlere nasıl uygulandığınıda gördükten sonra aklıma soyut sınıflara arayüzlerin uygulanması nasıl olur diye bir soru geldi. Biliyordumki arayüzleri bir sınıfa uyguladığımda, bildirilen metodları override etmeliyim. Soyut sınıflar arayüz bildirimindeki gibi metod bildirimleri dışında iş yapan metodlarda içerebiliyordu. Peki o halde, bir arayüzü bir soyut sınıfa uyguladığımda, arayüzdeki metod bildirimlerini bu soyut sınıfta override etmem gerekir miydi? Bunu anlamanın en iyi yolu arayüzü, soyut sınıflara uygularken metodları override

etmeyi ihmal etmeye çalışmaktı. Aynen aşağıdaki kodlarda olduğu gibi.

```
interface ITemel
{
   void Temel();
}

interface ITureyen1 extends ITemel
{
   void Tureyen1();
}

interface ITureyen2 extends ITureyen1
{
   void Tureyen2();
}

abstract class Soyut implements ITureyen1
{
   abstract public void Metod();
}
```

Yukarıdaki uygulamaya Soyut isminde bir abstract sınıf ekledim. Bu sınıfa ITureyen1 arayüzün uyguladım. Uygulamayı derlediğimde başarılı bir şekilde derlendiğini gördüm. Demek ki bir arayüzü bir soyut sınıfa uygularsam, arayüzdeki metod bildirimlerini bu soyut sınıf içinde override etmem gerekmiyormuş. Tabi elbetteki, Soyut isimli abstract sınıftan türetilen bir sınıfın hem bu soyut sınıftaki hemde bu soyut sınıfın uyguladığı arayüzlerdeki metodların hepsini override etmesi gerektiğini söylememede gerek yok.

Arayüzler ile ilgili en çok hoşuma giden özelliklerden biriside iç içe geçmiş (nested) arayüz tanımlamalarının yapılabilmesi. Yani bir arayüz tanımı başka bir arayüz tanımını ve hatta başka arayüz tanımlamalarını bünyesinde barındırabilmekte. Tek dikkat etmem gereken nokta, dahili arayüzlerin protected, private yada friendly erişim belirleyicilerine sahip olamamaları. Örneğin, aşağıdaki kodlar ile, içiçie geçmiş arayüzlerin bildirimi yapılıyor.

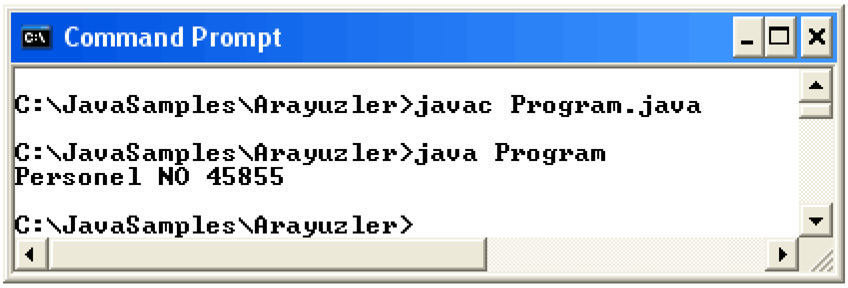
```java
interface IGenel
{
    interface IBilgi
    {
        void PersonelKunye(int ID);
    }

    interface IUcretlendirme
    {
        int MaasHesapla(int saatUcret,int saat);
    }

    interface IDepartman
    {
        String Kod(String departmanAd);
    }
}

class Personel implements IGenel.IBilgi
{
    public void PersonelKunye(int PerID)
    {
        System.out.println("Personel NO "+PerID);
    }
}

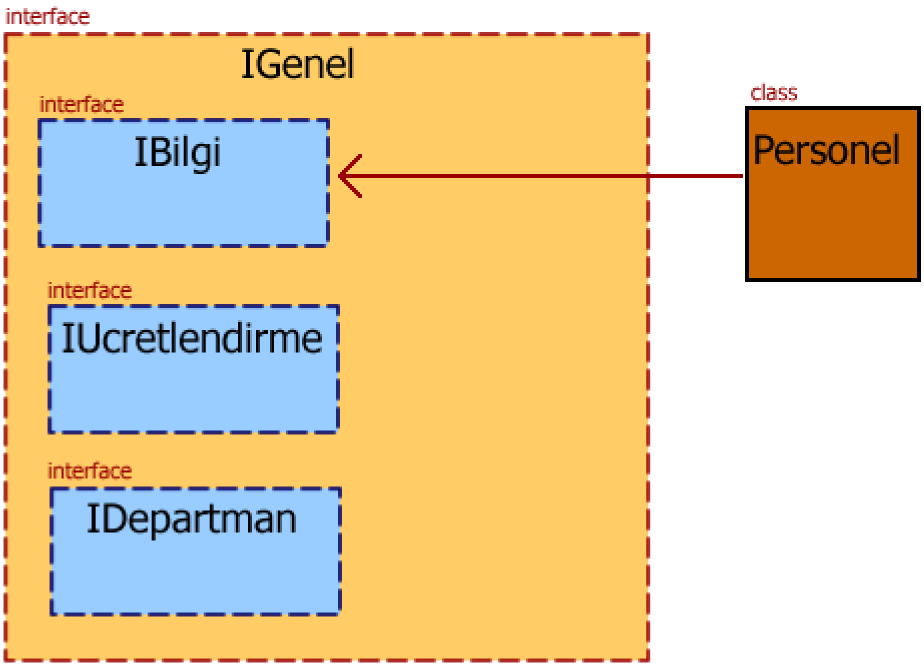
public class Program
{
    public static void main(String[] args)
    {
        Personel p=new Personel();
        p.PersonelKunye(45855);
    }
}
```



Burada görüldüğü gibi IGenel arayüzü 3 adet arayüz içermektedir. Sınıfımıza bu arayüzlerden sadece IBilgi isimli arayüzü uygulamak için aşağıdaki söz dizimini kullandım. Dolayısıyla Personel sınıfın için, IGenel arayüzü içindeki IBilgi arayüzünün uygulanacağını belirtmiş oldum.

```
class Personel implements IGenel.IBilgi
```

Burada olan şeyi şekil itibariyle ifade etmek gerekirse aşağıdaki grafiği gösterebilirim.



Elbette IGenel arayüzünün tamamınıda uygulayabiliriz. Bu durumda, IGenel arayüzü içerisinde yer alan tüm arayüzlerin içerdiği metod bildirimlerinin, ilgili sınıf tarafından override edilmeleri gerekmektedir. İç içe geçmiş arayüz tanımlamaları, birbirleriyle ilişkili, ortak çalışamalara ve amaçlara yönelik arayüzlerin bir çatı altında toplanmasında izlenebilecek etkili yollardan birisidir. Bu yapı aslında C# taki isim alanlarına veya javadaki paket kavramına benzer bir sistematikle işler.

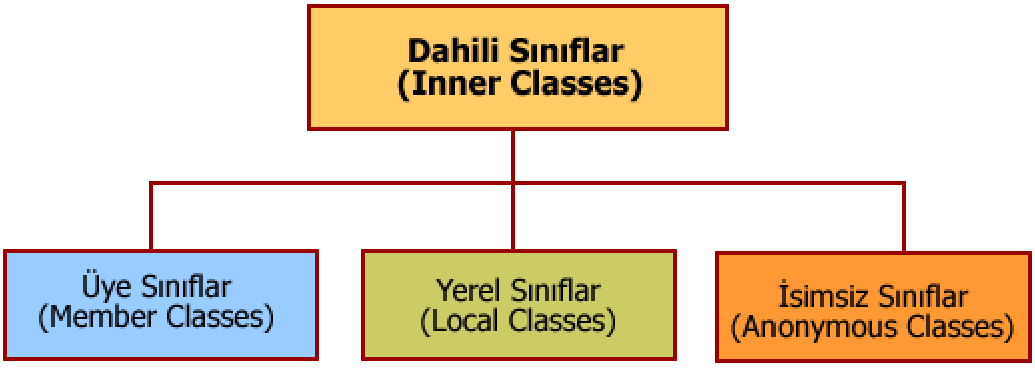
Arayüzler ile ilgili olarak kahvemizin söyleyeceği fazla bir şey yok aslında. Arayüzler faydalıdır. Dahası günlük uygulamalarımızda çok fazla başvuramadığımız bu sistematik yapılar, özellikle birden fazla programcının bir arada yer aldığı büyük çaplı uygulamalarda önem kazanmakta ve sınıfların oluşturduğu veri modellerine rehberlik yapmaktadır. Artık kafeinin etkisi giderek azalmaya ve göz kapaklarım kapanmaya başladı. Ancak bu kahve molası ile birlikte çok şey öğrendiğime inanıyorum. Bakalım gelecek kahve molalarında beni ne gibi sürpriz konular bekliyor.

Bölüm 12: Dahili Sınıflar (Inner Classes)

Bu hafta Java programlama dilinde kullanılan çok ilginç bir konuyu inceledim. Sınıflar içerisinde sınıfların tanımlanmasından tutunda, bir metod içinde sınıf tanımlanması gibi ilginç kavramları bünyesinde barındıran bir konuydu bu. Inner Class (Dahili Sınıflar) kavramı. Bu kahve molası açıkçası çok eğlenceli geçeceğe benziyor. Programatik olarak yani. Yoksa dışarıdan birisi bana bakıpta, yazdığım örnek kodlardan sonra hafifçe güldüğümü görse, sanıyorum ki bu adam keçileri kaçırmış der. Aslında hepimiz öyle değilmiyiz? Programcılar doğaları gereği hafif uçuk adamlar olmak zorundadırlar. Eğer kodlara bakıp kendi kendinize gülmek gibi bir hobiniz yoksa, hemen bir tane edinmenizi tavsiye ederim. Bir programcının karizmasını gerçekten önemli ölçüde arttıyor. Tabi gülmek derken sinsi sinsi gülüp hafifçede Platinyum Kadir Bey havası katacaksınız işin içine.

Neyse muhabbeti bırakıp iş yapmak sanırım daha hayırlı olucaktır. Nedir bu dahili sınıflar? Ne iş yaparlar? Neden kullanılırlar? Bu sorular içerisinde en zor olanı belkide dahili sınıflara neden ihtiyaç duyduğumuzdur. Doğruyu söylemek gerekirse, bu ihtiyacı anlayabilmek için, kaynaklarda kullanılan örnekleri bir hayli kurcalamam gerekti. İlk başta dahili sınıfların, sınıfları normal kullanım alanları haricinde farklı bölgelerde kullanmak için var olduklarını söyleyebilirim. Her zaman için uygulamalarımda şu ana kadar, sınıfları hep ayrı birer veri yapısı olarak tanımladım. Ancak aşağıdaki şekilde görülen dahili

sınıf tipleri ile, bir sınıfı başka bir sınıf bloğu içinde tanımlamak, bir metod bloğu içinde tanımlamak mümkün.



Özetle dahili sınıfların, sınıf tanımlamaları içerisinde tanımlanan sınıflar olarak düşünebiliriz. İlk önce üye sınıfları incelemeye karar verdim. Çünkü kaynaklar özellikle bu konuya oldukça fazla yer ayırmıştı. Üye sınıfları anlayabilmenin en iyi yolu basit bir örneğe uygulamaktan geçiyordu. Yapabileceğim en basit örnek, bir sınıf tanımlaması içine bir kaç sınıf tanımlaması eklemekti. Bu amaçla aşağıdaki gibi bir sınıf tanımlaması oluşturdum.

```
public class DahiliSiniflar
{
        class UyeSinif_1
        {

        }

        class UyeSinif_2
        {

        }

        class UyeSinif_3
        {

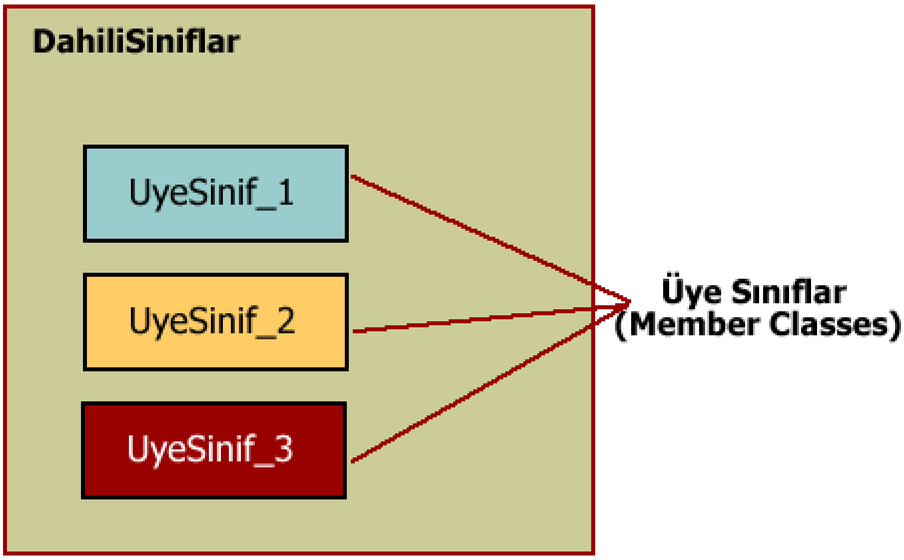
        }

    public static void main(String[] args)
```

```
    {

    }
}
```

Yukarıdaki uygulamayı derlediğimde herhangibir hata mesajı ile karşılaşmadım. Yani DahiliSiniflar sınıfım içindeki diğer sınıflar için derleyici bana ses çıkartmamıştı. İşte burada yaptığım iş, aslında bir sınıf içerisine, bu sınıfın kapsüllediği Üye Sınıf (Member Class) 'ları eklemekti. Bir başka deyişle aslında aşağıda şekilsel olarak ifade edebildiğimi düşündüğüm işlemi gerçekleştirmiştim.



Her şey iyiydi güzeldi de, acaba bu üye sınıfları nasıl kullanabilirdim. Öncelikle, üye sınıflara iş yapacak metodlar eklemeliydim. Bu amaçla kodları aşağıdaki gibi düzenledim. İşin kritik noktası, üye sınıf nesnelerinin, bu üye sınıfları içeren sınıfa ait main metodu içerisinde nasıl örneklendirildikleriydi.

```
public class DahiliSiniflar
{
        class UyeSinif_1
        {
            void Yaz()
            {
                System.out.println("UyeSinif_1 Yaz Metodu...");
            }
```

```
    }

    class UyeSinif_2
    {
      void Alan(double en, double boy)
      {
        double alan=en*boy;
        System.out.println("Alan "+alan);
      }
    }

    class UyeSinif_3
    {
      int Topla(int baslangic, int bitis)
      {
        int toplamDeger=0;

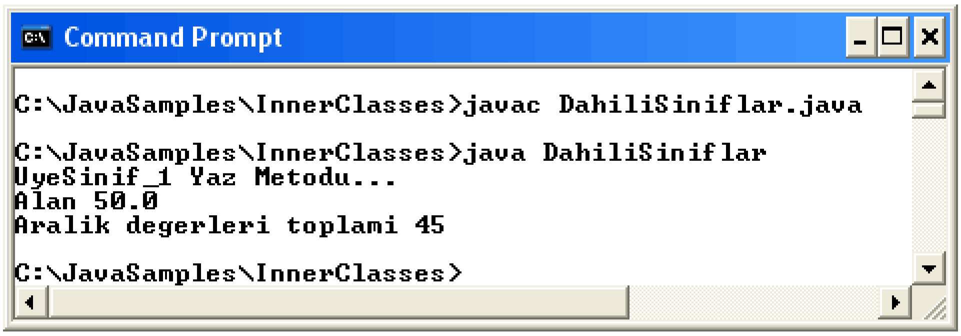
        for(int i=baslangic;i<=bitis;i++)
        {
          toplamDeger+=i;
        }
        return toplamDeger;
      }
    }

  public static void main(String[] args)
  {
    DahiliSiniflar.UyeSinif_1 uye1=new DahiliSiniflar().new
UyeSinif_1();
    uye1.Yaz();

    DahiliSiniflar.UyeSinif_2 uye2=new DahiliSiniflar().new
UyeSinif_2();
    uye2.Alan(5,10);

    DahiliSiniflar.UyeSinif_3 uye3=new DahiliSiniflar().new
UyeSinif_3();
    int sonuc=uye3.Topla(5,10);
    System.out.println("Aralik degerleri toplami "+sonuc);
  }
```

```
}
```



Burada önemli olan nokta, üye sınıflara ait bir nesne örneği yaratabilmek için, önce üye sınıfı kapsayan sınıfa ait bir nesnenin oluşturulmak zorunda olduğu. Yani,

```
DahiliSiniflar.UyeSinif_1 uye1=new DahiliSiniflar().new
UyeSinif_1();
```

satırları ile, önce DahiliSiniflar sınıfından bir nesne oluşturuluyor ve ardından bu sınıfın üye sınıfı olan dahili UyeSinif_1 sınıfına ait bir nesne örneği oluşturuluyor. Evet söz dizimi oldukça garip, bunu bende kabul ediyorum. Ancak, olayı mantık boyutları çerçevesinde son derece güzel açıklıyor. Tabi burada kafama takılan en önemli sorun şu. Her üye sınıf nesnesi örneği için, üye sınıfı içeren sınıf örneğini oluşturmak zorundamıyım. Bu durumda n kadar üye sınıf içeren bir sınıf uygulamasında, her bir üye sınıf için bellekte gereksiz yer işgali olmıyacakmı? Yazık değilmi güzelim sistem kaynaklarını harcamaya?

Neyseki bu durumun çözümünü kısa sürede kaynaklardan tedarik ettim. Çözüm ilk başta, üye sınıfları içeren sınıfa ait bir nesne örneğini oluşturmak ve daha sonra üye sınıflara ait nesne örnekleri için, oluşturulan bu sınıf örneğinin referansını kullanmaktı. Yani şu şekilde,

```
public static void main(String[] args)
{

    DahiliSiniflar ds=new DahiliSiniflar();

    DahiliSiniflar.UyeSinif_1 uye1=ds.new UyeSinif_1();
```

```
    uye1.Yaz();

    DahiliSiniflar.UyeSinif_2 uye2=ds.new UyeSinif_2();
    uye2.Alan(5,10);

    DahiliSiniflar.UyeSinif_3 uye3=ds.new UyeSinif_3();
    int sonuc=uye3.Topla(5,10);
    System.out.println("Aralik degerleri toplami "+sonuc);

}
```

Şimdi kafamı kurcalayan nokta, üye sınıf yapıcıları ile, üye sınıfları içeren sınıf yapıcısı arasındaki ilişkinin içeriğiydi. Sonuçta ya kapsül sınıfın tek bir nesne örneğine başvurarak, üye sınıf nesneleri oluşturuyor yada her üye sınıf nesnesi için birer kapsül sınıf nesnesi oluşturuluyordu. Her ne şekilde olursa olsun, kapsül sınıfın yapıcısınında, üye sınıfın yapıcısınında çalıştırılacağı kesindi. Peki sıralama nasıl olucaktı? Elbette burada kalıtımda yapıcıların nasıl çalıştığı göz önüne alınabilirdi. Kalıtıma göre, en üst ata sınıfın yapıcısı ilk olarak çağırılacak, daha sonra hiyerarşide aşağıya doğru inilerek diğer yapıcılar sırasıyla işleyecekti. Bu işlevi izleyebilmek amacıyla aşağıdaki örneği geliştirdim.(Her zamanki gibi iş yapmayan ama amacı kavramaya yönelik bir örnek...)

```
public class Uygulama
{

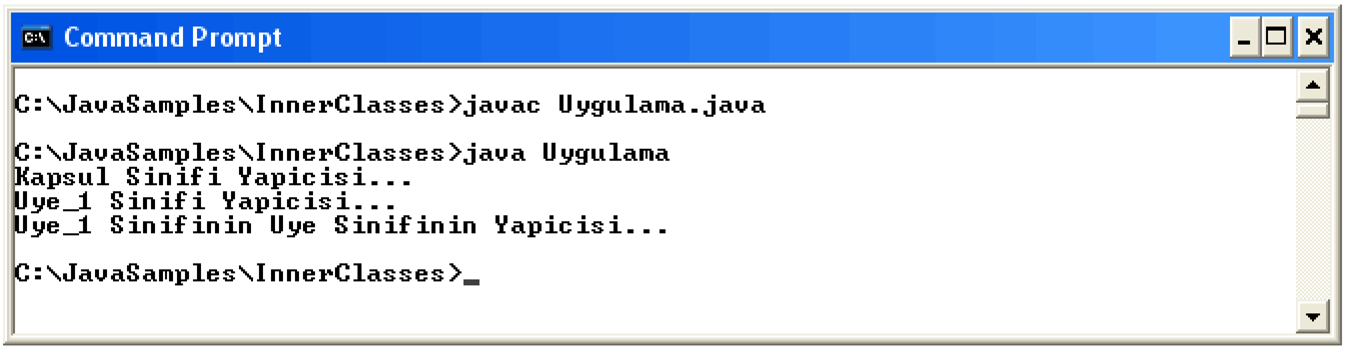
    class Uye_1
    {
        Uye_1()
        {
        System.out.println("Uye_1 Sinifi Yapicisi...");
        }

            class Uye_1_AltUye
            {
                Uye_1_AltUye()
                {
                    System.out.println("Uye_1 Sinifinin Uye Sinifinin
```

```
Yapicisi...");
                }
            }
        }

    Uygulama()
    {
        System.out.println("Kapsul Sinifi Yapicisi...");
    }

    public static void main(String[] args)
    {
        Uygulama.Uye_1.Uye_1_AltUye nesne=new
Uygulama().new Uye_1().new Uye_1_AltUye();
    }
}
```



Her ne kadar, en alttaki üye sınıfa ait nesneye oluşturmak uzun ve zahmetli bir yol olarak gözüksede sonuç itibariyle amacıma ulaşmıştım. Yapıcılar tahmin ettiğim gibi kapsülleyen sınıftan çalıştırılmaya başlamıştı. Zaten bu, Uye_1_AltUye üye sınıfına ait nesnesnin oluşturulmasından önce diğer üst sınıf nesnelerinin oluşturulma zorunluluğunun bir sonucuydu.

Üye sınıflar ile ilgili bir diğer ilginç nokta, üye sınıflarında public ve friendly dışında, protected ve private gibi erişim belirleyicilerinin kullanılabilmesiydi. Normal şartlar altında, bir sınıfı private yada protected erişim belirleyicileri ile tanımlamaya çalışmak olanaksızdı. Oysaki üye sınıflarda durum farklıydı. Örneğin bir üye sınıfı private yaparak, kapsül sınıfı dışındaki diğer sınıflarca kullanılması engellenebilirdi. Nasıl mı? İşte örneği.

```
class DahiliSiniflar
{
    public class UyeSinif_1
    {
        void Yaz()
        {
            System.out.println("UyeSinif_1 Yaz Metodu...");
        }
    }

    protected class UyeSinif_2
    {
        void Alan(double en, double boy)
        {
            double alan=en*boy;
            System.out.println("Alan "+alan);
        }
    }

    private class UyeSinif_3
    {
        int Topla(int baslangic, int bitis)
        {
            int toplamDeger=0;

            for(int i=baslangic;i<=bitis;i++)
            {
                toplamDeger+=i;
            }
            return toplamDeger;
        }
    }

    class UyeSinif_4
    {
        UyeSinif_4()
        {
            System.out.println("UyeSinif_4 yapicisi");
        }
    }
```

```
}

public class Uygulama2
{
    public static void main(String[] args)
    {
        DahiliSiniflar ds=new DahiliSiniflar();

        DahiliSiniflar.UyeSinif_1 uye1=ds.new UyeSinif_1();
        uye1.Yaz();

        DahiliSiniflar.UyeSinif_2 uye2=ds.new UyeSinif_2();
        uye2.Alan(5,10);

        DahiliSiniflar.UyeSinif_3 uye3=ds.new UyeSinif_3();
        int sonuc=uye3.Topla(5,10);
        System.out.println("Aralik degerleri toplami "+sonuc);

        DahiliSiniflar.UyeSinif_4 uye4=ds.new UyeSinif_4();
    }
}
```

Bu örneğin bukadar uzun olduğuna bakmamak lazım. Odaklanılması gereken nokta, private üye sınıfın, tanımlandığı kapsül sınıfı dışındaki bir sınıf içerisinde örneklendirilmeye çalışılmasıdır. Uygulamayı derlediğimde aşağıdaki hata mesajlarını anladım.

```
Command Prompt
C:\JavaSamples\InnerClasses>javac Uygulama2.java
Uygulama2.java:56: DahiliSiniflar.UyeSinif_3 has private access in DahiliSinifla
r
                DahiliSiniflar.UyeSinif_3 uye3=ds.new UyeSinif_3();
                              ^
Uygulama2.java:56: DahiliSiniflar.UyeSinif_3 has private access in DahiliSinifla
r
                DahiliSiniflar.UyeSinif_3 uye3=ds.new UyeSinif_3();
                                                      ^
Uygulama2.java:56: UyeSinif_3() has private access in DahiliSiniflar.UyeSinif_3
                DahiliSiniflar.UyeSinif_3 uye3=ds.new UyeSinif_3();
                                                  ^
3 errors

C:\JavaSamples\InnerClasses>_
```

Sonuç gayet açık ve netti. Private tanımlanmış bir üye sınıfa, tanımlandığı kapsül sınıf dışından erişememiştim. Ancak bu

kısıtlama üye sınıfa, tanımlandığı kapsül sınıf içindeki başka üye sınıflar tarafından erişilemiyeceği anlamına gelmemekteydi. Dolayısıyla, private üye sınıfa ait nesne örneğini, başka bir üye sınıf içinde aşağıda olduğu gibi kullanabilirdim. Tek şart üyelerin, aynı kapsül sınıf içinde tanımlanmış olmalarıydı.

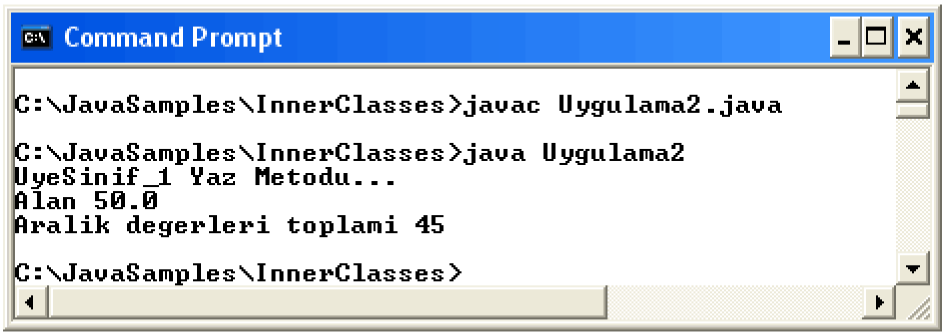
```
class DahiliSiniflar
{
...
    private class UyeSinif_3
    {
        int Topla(int baslangic, int bitis)
        {
            int toplamDeger=0;

            for(int i=baslangic;i<=bitis;i++)
            {
                toplamDeger+=i;
            }
            return toplamDeger;
        }
    }

    class UyeSinif_4
    {
        UyeSinif_4()
        {
            DahiliSiniflar.UyeSinif_3 uye3=new DahiliSiniflar().new
UyeSinif_3();
            int sonuc=uye3.Topla(5,10);
            System.out.println("Aralik degerleri toplami "+sonuc);
        }
    }
}

public class Uygulama2
{
    public static void main(String[] args)
    {
        ...
```

```
        DahiliSiniflar.UyeSinif_4 uye4=ds.new UyeSinif_4();
    }
}
```



Artık Bonus Plus'a geçmenin vakti gelmişti. Vitesi arttırıp konuda ilerlemeye devam etmeliydim. Üye sınıflara burada nokta koyup diğer dahili sınıfları incelemem gerektiği düşüncesindeydim. Ancak yapamadım. İncelemem gereken bir konu daha olduğunu farkettim. Static üye sınıflar. Nasıl ki static metodları çalıştırmak için nesne örneklerine gerek yoksa, static üye sınıfları oluşturmak içinde, kapsül sınıfın nesne örneğini oluşturmaya gerek yoktur diye düşündüm ilk olarak. Acaba durum böyle miydi? İlk önce static bir üye sınıf tanımlamakla işe başladım. Sonraki amacım bu static sınıfa ait bir nesne örneğini oluşturmaktı. Bu nesne örneğini ilk başta klasik üye sınıfı örneklendirme tekniğiyle oluşturmaya çalıştım.

```
public class Uygulama
{
    static class Static_Uye
    {
        Static_Uye()
        {
            System.out.println("Statik_Uye Sinifi Yapicisi...");
        }
    }

    public static void main(String[] args)
    {
        Uygulama.Static_Uye uye=new Uygulama().new
Static_Uye();
    }
```

```
}
```

Kod başarılı bir şekilde derlenmişti. Şimdi main yordamı içindeki nesne örneklendirme satırını aşağıdaki gibi değiştirdim.

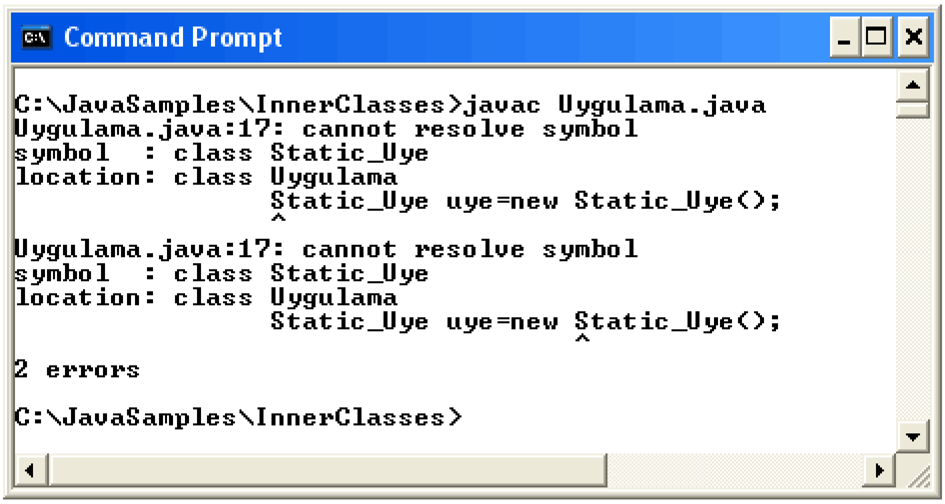
```
Static_Uye uye=new Static_Uye();
```

Kod yine başarılı bir şekilde derlendi ve çalıştı. İki teknik arasında oldukça önemli bir fark vardı. İkinci kullanımda bu üye sınıfı static olarak tanımladığım için, kapsül sınıftan bir nesne oluşturma zahmetine girmemiştim. Dolayısıyla kapsül sınıf içinde bu üye sınıfı normal bir sınıf türetirmişçesine örnekleyebilmiştim. Peki durum kapsül sınıf dışındaki sınıflar için nasıl ceyran edecekti. Bunun için yapmam gereken basit bir örnek ile durumu aydınlatmaya çalışmaktı.

```
class Deneme
{
    static class Static_Uye
    {
        Static_Uye()
        {
            System.out.println("Statik_Uye Sinifi Yapicisi...");
        }
    }
}

public class Uygulama
{
    public static void main(String[] args)
    {
        Static_Uye uye=new Static_Uye();
    }
}
```

Uygulamayı derlediğimde aşağıdaki hata mesajları ile karşılaştım.



Çözüm gayet basitti. Statik sınıfa ait nesne örneğini oluştururken, statik sınıfın tanımlı olduğu kapsül sınıfıda kullanmalıydım. Nasıl ki, işte şöyle ki.

```
Deneme.Static_Uye uye=new Deneme.Static_Uye();
```

Bu haliyle uygulama başarılı bir şekilde derlendi. Ancak yine dikkat edilmesi gereken nokta, kapsül sınıfa ait bir nesne örneğinin oluşturulmamış olmasıydı. Bununla birlikte static üyelerin, kapsül sınıftaki alanlara ve diğer üyelere erişiminde bir sorun olduğu kaynaklarımda geçiyordu. Bunu en güzel bir örnekle irdeleyebileceğimi biliyordum.

```
class Deneme
{
    String alan="BURAK";

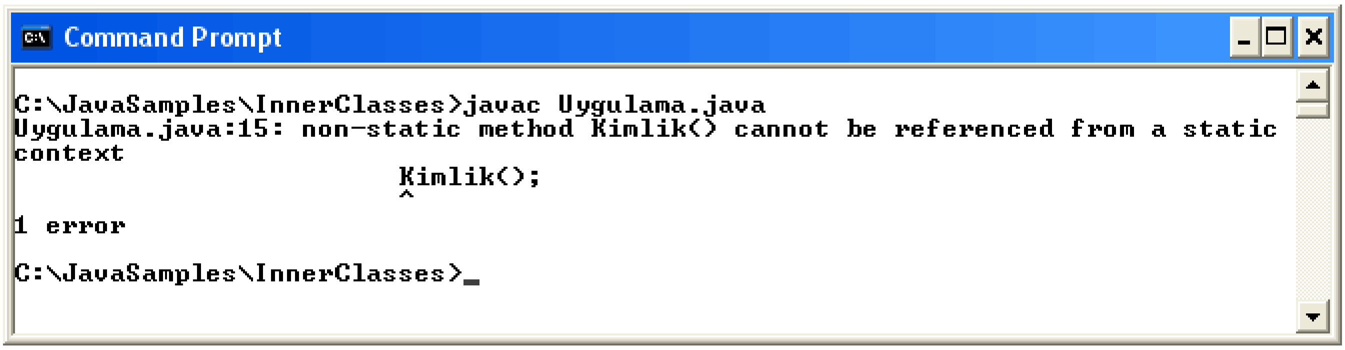
    void Kimlik()
    {
        System.out.println("MERHABA... "+alan);
    }

    static class Static_Uye
    {
        Static_Uye()
        {
            System.out.println("Statik_Uye Sinifi Yapicisi...");
            Kimlik();
        }
```

```
    }
}

public class Uygulama
{
    public static void main(String[] args)
    {
        Deneme.Static_Uye uye=new Deneme.Static_Uye();
    }
}
```

Bu uygulamayı derlediğimde aşağıdaki hata mesajını aldım.



Bunun sebebini kaynaklar şu şekilde açıklıyor. Static bir üye sınıfın, kaspül sınıf ile arasında this ilişkisi yoktur. Buradan çıkartabileceğim sonuç, static üye sınıfın kendi içinden, kapsül sınıftaki üyelere (alanlar ve metodlar) erişemiyeceğiydi. Nitekim, buradaki sınıfımız statik olmasaydı, kapsül sınıftaki üyelere erişebilirdi. Ancak durum kapsül sınıftaki üyelerde statik olursa değişmekteydi. Yani örneği aşağıdaki şekilde geliştirdiğimde, uygulama başarılı bir şekilde derleniyordu. Çünkü üye sınıf, kapsül sınıfın nesne örneğine ihtiyaç duyulmadan kullanılabilecek static üyelere erişmekteydi.

```
class Deneme
{
    static String alan="BURAK";

    static void Kimlik()
    {
        System.out.println("MERHABA... "+alan);
    }
```

```
    static class Static_Uye
    {
        Static_Uye()
        {
            System.out.println("Statik_Uye Sinifi Yapicisi...");
            Kimlik();
        }
    }
}
```

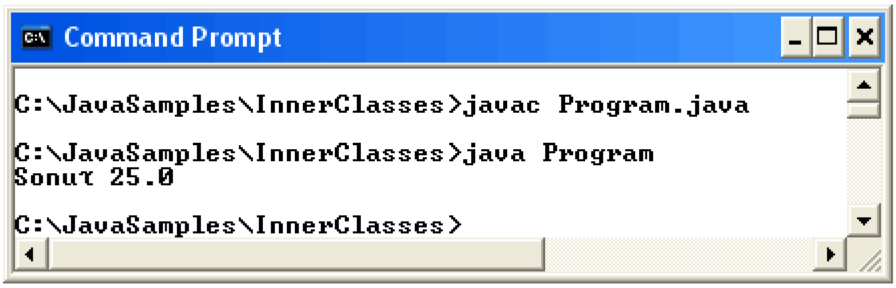
Artık sıra yerel sınıfları öğrenmeye geldi sanıyorum. Ama bu dahili sınıf tipine geçmeden önce özellikle, üye sınıfların genel kullanım amacı üzerinde durmakta fayda olacağını düşünüyorum. Üye sınıflar, çoklu kalıtımın gerçekleştirilmesine imkan sağlamaktadırlar. Nitekim, bir sınıfı bir kaç sınıftan birden türetmek mümkün değildir. Ancak bir sınıfa bir kaç arayüz uygulanarak çoklu kalıtım kısmide olsa sınırlı bir şekilde uygulanabilir. Bununla birlikte üye sınıflar ile çoklu kalıtım daha güvenli ve güçlü bir şekilde gerçekleştirilebilir. Bu amaçla geliştirdiğim aşağıdaki basit örneği incelemekte fayda var sanırım.

```
class TemelSinif
{
    void Yaz(double deger)
    {
        System.out.println("Sonuc "+deger);
    }
}

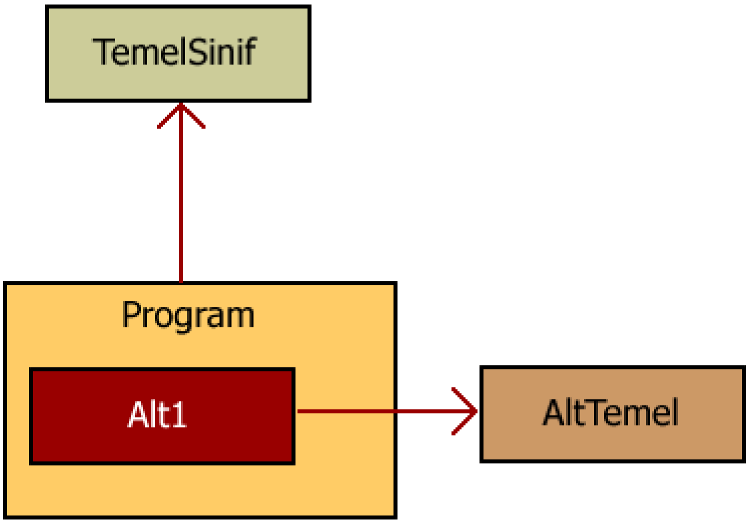
class AltTemel
{
    double Alan(double en, double boy)
    {
        return (en*boy)/2;
    }
}
public class Program extends TemelSinif
{
```

```
    class Alt1 extends AltTemel
    {
        Alt1()
        {
            Yaz(Alan(20,2.5));
        }
    }

    public static void main(String[] args)
    {
        Program.Alt1 nesne=new Program().new Alt1();
    }
}
```



Peki burada olan olay nedir? Çoklu kalıtım nerede gizlidir? Bunu cevaplandırabilmenin en kolay yolu, uygulamadaki sınıflar arasındaki ilişkiyi şematik olarak ifade edebilmektir. Bu amaçla, uml vari bir diagam geliştirmeye çalıştım.



Şekilde herşey daha net. (.net değil berrak anlamında...) Program sınıfı içerisinde yer alan Alt1 üye sınıfı, AltTemel isimli

sınıftan türetiliyor. Lakin, üye sınıfımızın bulunduğu kapsül sınıf olan Program sınıfıda, TemelSinif'tan türetilmiştir. Dolayısıyla burada, Alt1 sınıfı hem AltTemel, hemde TemelSinif sınıflarının üyelerine erişim hakkına kalıtım gereği sahiptir. Zaten örnektede bunu irdelemeye çalıştım. Üye sınıf yapcısı içinden, üye sınıfın içinde bulunduğu kapsül sınıfın türediği temel sınıftaki metodu, üye sınıfın türediği sınıfa ait metodu parametre alacak şekilde çağırdım. İşte çok biçimlilik denen olayın üye sınıflar ile güvenli bir biçimde uygulanabilmesi.

Sanıyorumki artık diğer dahili sınıflara geçebilirim. Bu umutla kaynaklarıma son kez bakıp titreyen eller ile, umarım üye sınıfların başka sürprizleri yoktur diyerek sayfalarda ilerlemeye devam ettim. Ancak biliyorumki üye sınıflar ile ilgili ileride karşıma yeni şeyler çıkacak. Bu java dilinin doğasında olan bir şey. Sürekli sürprizler yapıyor bu çılgın programlama dili. Giderek sevmeye mi başladım ne.

Nihayet!!! Sonunda yerel sınıflara gelebildim. Yerel sınıflar, üye sınıflardan daha ilginç bir kavram. Öyleki, bir metodun içersinde kullanılmak üzere sınıf bildirimleri gerçekleştirilebiliyor. Sadece metod olsa iyi. Yapıcılar içindede yerel sınıf tanımlamları yapılabilmekte. Ancak burada önemli olan yerel sınıfın sadece tanımlandığı metod bloğu içerisinde çalışıyor olması. Dolayısıyla bu metod dışından bu sınıflara erişmek mümkün değil. Neden böyle bir gereksinimim olabilir sorusuna gelince, bu önümüzdeki aylar boyunca kendi kendime soracağım bir soru olarak kalıcak sanıyorum. Ama bu tarz sorulara verilen güzel bir cevabıda görmezden gelemem. "Dilin kabiliyetlerini bilmemizi sağlar..." Bir bakalım yerel sınıflar neyin nesiymiş.

```
public class Aritmetik
{
    static double Hesaplamalar(double a,double b)
    {

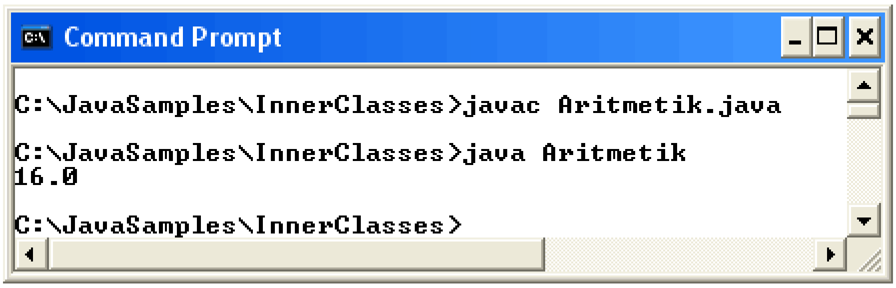
        class Alan
        {
            double en;
            double yukseklik;
```

```
            Alan(double deger1,double deger2)
            {
                en=deger1;
                yukseklik=deger2;
            }

            public double Hesapla()
            {
                return (en*yukseklik)/2;
            }
        }

        Alan nesne=new Alan(a,b);
        return nesne.Hesapla();
    }

    public static void main(String[] args)
    {
        double sonuc=Hesaplamalar(8,4);
        System.out.println(sonuc);
    }
}
```



Olay son derece açıktı aslında. Metod içinde bir yerel sınıf tanımlanmış ve kullanılmıştı. İşte yerel sınıfların özü buydu. Yerel sınıflar, kullanıldıkları metod dışında işlevsel değildi. Bununla birlikte, bir yerel sınıfı başka sınıflardan veya arayüzlerden türetebilirdik. Yerel sınıflar ile ilgili önemli bir kısıtlamada, sadece friendly erişim belirleyicisine sahip olabilmesiydi. Yani üye sınıflarda olduğu gibi tüm erişim belirleyicilerini kullanılamıyorlar. Zaten, sadece metod içinde geçerli olmasınıda bu durum bir nebze olsun açıklıyor.

Kahvem azaldıkça gecenin bu geç saatlerinde enerjimde azalmaya başladı. Ancak işlemem gereken bir dahili sınıf çeşidi daha var. Bu dahili sınıf türü, isimsiz sınıflar. Açıkçası bu sınıfların kullanımını düşündükçe biraz güldüm kendimce. Nasıl mı? Haydi kahvemizden bir yudum daha alalım ve isimsiz sınıfları inceleyelim.

```
interface Alan
{
    public double Hesapla();
}

public class Aritmetik2
{
    public static Alan Hesaplamalar(final double a,final double b)
    {

        return new Alan()
        {
            public double Hesapla()
            {
                return (a*b)/2;
            }
        };
    }

    public static void main(String[] args)
    {
        Alan a=Hesaplamalar(8,4);
        double sonuc=a.Hesapla();
        System.out.println(sonuc);
    }
}
```

Neden güldüğümü şimdi daha iyi anlamışsınızdır herhalde. Kodu kaynaklarımdaki örnekleri inceleyerek oluşturdum. Sonrada, örneğimin içinde, isimsiz sınıfı aramaya başladım. İşte o anda bir gülme krizi aldı beni gitti. İsimsiz sınıfı bulamıyordum. Kod her zamankinden farklıydı, hemde çok farklıydı ancak ben isimsiz sınıfı bir türlü bulamıyordum. Sonradan dikkatimi, Hesaplamalar isimli metodunun geri dönüş değerinin türü çekti.

Bu metod dönüş tipi olarak bir arayüzü nesnesi işaret etmekteydi. Bir arayüz nesnesi geri döndürecekse eğer, return anahtar kelimesinden sonra bu arayüzü uygulayan bir sınıf örneğinin dönmesi gerektiğini anlamıştım.

İşte isimsiz sınıfım oradaydı. Hakikattende isimsiz bir sınıftı. Adı yoktu ama şanı almış başını yürüyordu. Gözükmeyen kahraman gibiydi desem yeridir. return anahtar kelimesinden sonra, metodun dönüş tipi olan arayüzü'e ait bir sınıf oluşturulmuş ve içinde bir takım hesaplamalar yaptırılmıştı. Yani isimsiz sınıfım şurasıydı.

```
return new Alan()
{
      public double Hesapla()
      {
         return (a*b)/2;
      }
};
```

Açıkçası anlaşılması ve kullanımı oldukça karmaşık bir yapıda. Tam olarak kavrayabildiğimi söyleyemem. Ancak kaynaklarım özellikle ileride göreceğim, olay dinleyicileri (event listener) ile ilgili yerlerde sıklıkla kullanıldığını belirtiyor. En azından şimdilik, nasıl kullanıldığını gördüm. Zaman ilerledikçe bu kavramında yerine oturacağı kanısındayım. Ancak şimdi dinlenme zamanı geldi diye düşünüyorum. Kahvemde bitti zaten.

Bölüm 13: İstisnalar (Exceptions)

Nesne yönelimli programlama dillerini özel kılan yanlardan biriside sağlamış oldukları istisna yönetim mekanizmalarının çeşitliliğidir. Çoğu zaman, özellikle çalışma zamanında oluşabilecek hataların gölgesinden kurtulmak ve kullanıcı dostu projeler geliştirmek zorundayız. Bunu yaparken sadece doğru kurallar ve algoritmalar yeterli olmayabilir. Özellikle kullanıcı

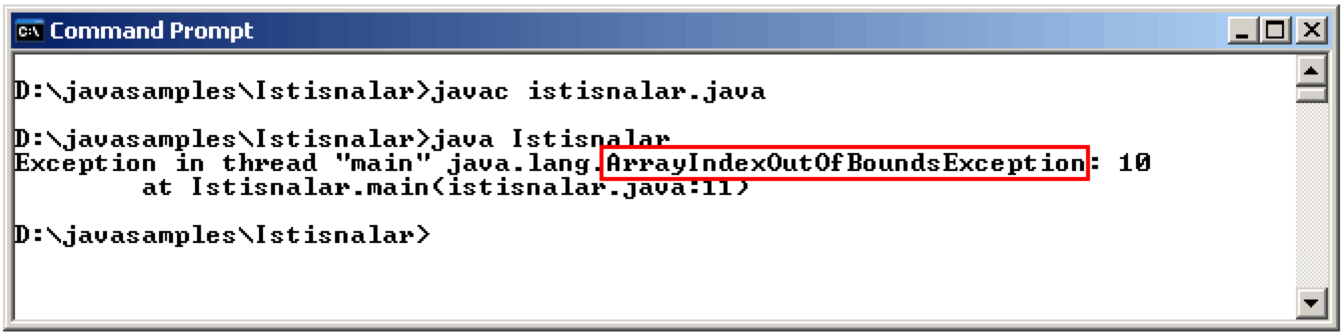
etkileşimli tasarımlarda, gelebilecek değişik ve vurdumduymaz taleplere karşı hazırlıklı olmak gerekir.

Java dilinde olsun C# dilinde olsun, çalışma zamanında oluşabilecek yada derleme zamanında oluşması kesin bir takım hataların önüne geçmek amacıyla istisna mekanizmaları kullanılmaktadır. Istisna mekanizlamaları şu meşhur herkesin duyduğu hatta bildiği try-catch-finally bloklarının kullanıldığı teknik ile sağlanıyor. Kanımca, istisnaları anlayabilmenin en iyi yolu istisna fırlatan (ki bu terim throwing olarak geçiyor) bir uygulama geliştirmekten geçiyor. Bu amaçla aşağıdaki gibi çok basit bir uygulama tasarladım. Oluşacak hata baştan belliydi. Balık baştan kokmuştu. Ama neyseki keskin Jacobs aromalı kahvem bu kokuyu bastırıyordu. Kalkıpta, 10 elemanlı bir dizinin 10nuci indeksine, bu dizinin 0 tabanlı olduğunu bile bile erişmeye çalışmak elbette bellekte böyle bir yer ayrılmadığı için hataya neden olucaktı. Ama istisnaları ele almanında en kolay yolu böyle basit bir örnekten geçiyordu.

```
public class Istisnalar
{
    public static void main(String[] args)
    {
        int dizi[]=new int[10];

        for(int i=0;i<10;i++)
        {
            dizi[i]=i*i;
        }
        System.out.println("Dizinin 10ncu elemani "+dizi[10]);
        System.out.println("Program sonu");
    }
}
```

Bu uygulamayı derleyip çalıştırdığımda aşağıdaki gibi bir hata mesajı ile karşılaştım. Aslında şu ana kadarki kahve molalarımda, çoğu zaman hatalar ile karşılaşmıştım ve durumu düzeltmiştim. Ancak bu kez meydana gelen bu hatanın çalışma zamanından nasıl önleneceğini ve uygulamanın aniden sonlandırılmasının nasıl önüne geçileceğini inceleyecektim.



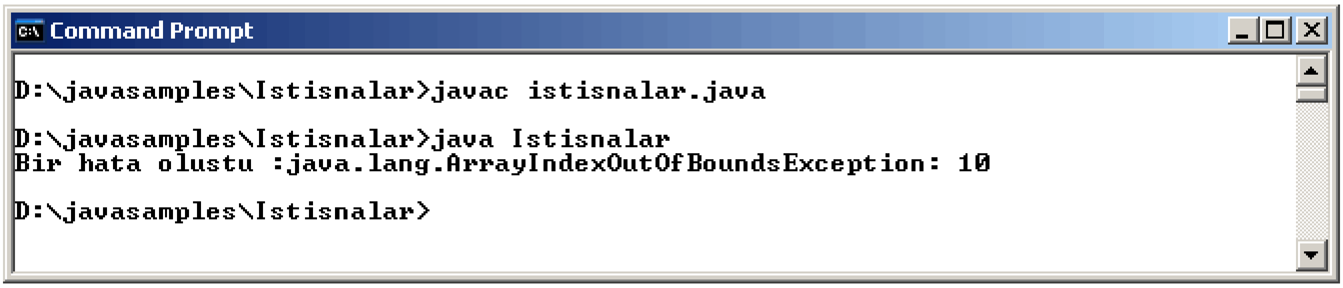
Burada gerçekleşen en önemli şey, programın çalışması sırasında bir istisnanın oluşması ve sonlandırılmasıdır. Öyleki, hatanın meydana geldiği satırdan sonraki program kodu ifadesi çalıştırılmamıştır. Yani, isteğimiz dışında bir sonlandırılma gerçekleştirilmiştir. Buda elbetteki istenmeyen bir durumdur. Ancak, istisna yönetim mekanizması yardımıyla bu kodun sorunsuz bir şekilde çalışmasını ve istemsiz olarak aniden sonlandırılmasını engelleme şansına sahibim. Nasıl olucak bu iş? Elbetteki, C# dilinden gelen engin tecrübelerimi burada aynen değerlendirmek taraftarıyım. Kodu şu şekilde düzenlersem istediğimi elde edeceğimi biliyordum.

```
public class Istisnalar
{
    public static void main(String[] args)
    {
        int dizi[]=new int[10];

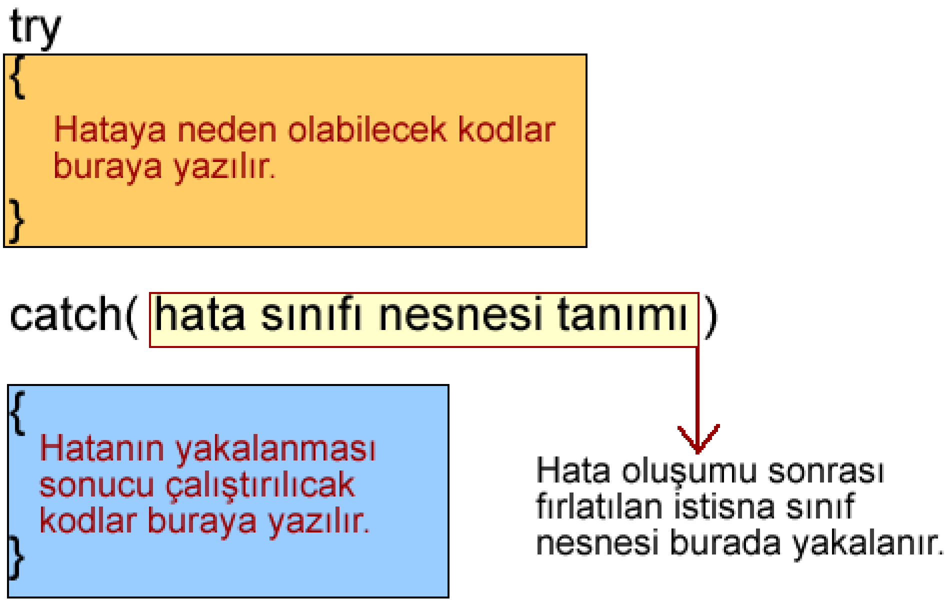
        for(int i=0;i<10;i++)
        {
            dizi[i]=i*i;
        }

        try
        {
            System.out.println("Dizinin 10ncu elemani "+dizi[10]);
            System.out.println("Program sonu");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Bir hata olustu :"+e);
```

```
        }
    }
}
```



Burada yapılan işlem aslında try-catch blokları ile hata takibinin en basit şekliydi. Şematik olarak durum ancak aşağıdaki kadar güzel bir şekilde dramatize edilebilirdi.



İki uygulamanın sonucu arasında çok fazla fark yokmuş gibi görünüyor ilk başta. Acaba gerçekten böyle mi? Elbette değil. Herşeyden önce, bu kez hataya yol açan kod satırını try bloğu içerisine aldım. Diğer yandan, oluşacak olan hatayı catch bloğunda yakalayarak kendi istediğim kod satırının devereye girmesini sağladım. Bu oldukça önemli. Nitekim, programın çalışmasının normalde sonlandırılacağı yerde, catch bloğundaki kodların devreye girmesini sağlamış oldum. Bu, özellikle kullanıcı taraflı hataların değerlendirilmesinde oldukça önemlidir. Ancak bu kod satırlarında dikkat çeken diğer bir nokta, hatanın bir nesne olarak tanımlanması ve mesaj olarak gösterilmek için kullanılmasıdır.

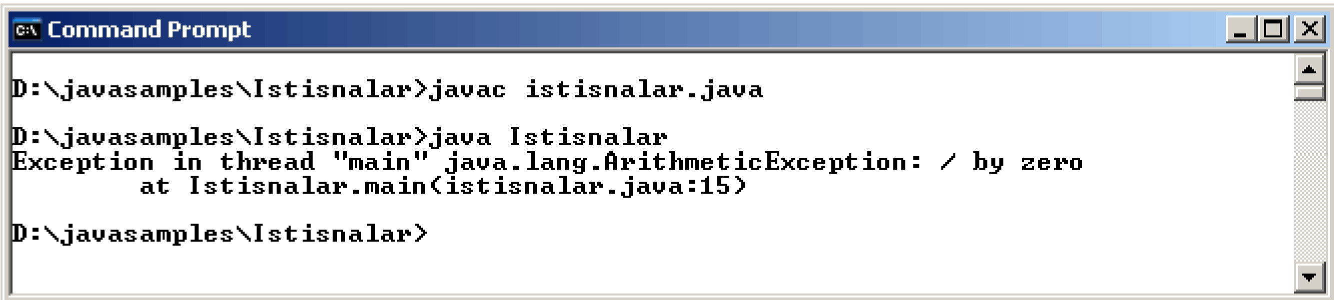
```
catch(ArrayIndexOutOfBoundsException e)
```

Aslında burada yapılan, derleme zamanında try bloğu altında meydana gelecek hatalarda, catch bloğunda belirtilen sınıf nesnesinin işaret ettiği türden bir istisnanın fırlatılmasıdır. Dolayısıyla, burada belirtilen türden bir istisna oluşursa ne olucaktır? Dizinin boyutu dışındaki bir indekse erişilmesi sonucu oluşturulan bu istisna nesnesi son derece spesifiktir. Ancak durum her zaman bu şekilde olmayabilir. Herşeyden önce, bunu irdelemenin en iyi yolu aynı program kodunda farklı türden bir hatanın meydana gelmesine yol açmaya çalışmak olmalıdır. Şöyleki,

```
try
{
    int a=1;
    int b=a/0;

    System.out.println("Dizinin 10ncu elemani "+dizi[10]);
    System.out.println("Program sonu");
}
catch(ArrayIndexOutOfBoundsException e)
{
    System.out.println("Bir hata olustu :"+e);
}
```

Bu haliyle kodları çalıştırdığımda, aşağıdaki hata mesajı ile uygulamanın şahane bir şekilde sonlandırıldığını gördüm. Ne yazdığım catch bloğı devreye girmiş nede uygulama istediğim şekilde sonlanmıştı.

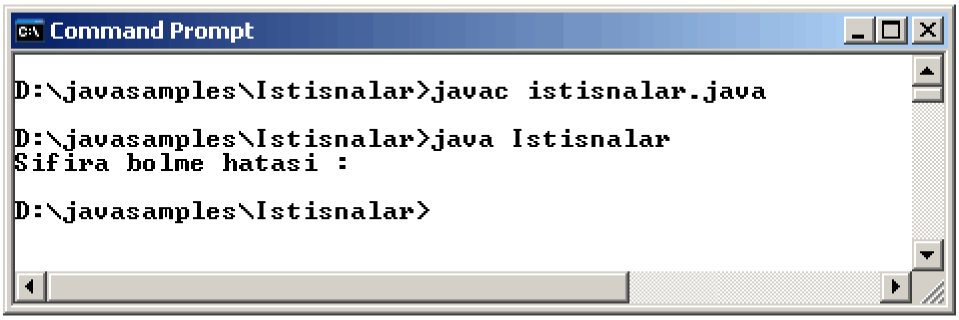


Sorun şuydu. Catch bloğunda sadece, indeks taşmasına ilişkin bir istisna nesnesi yakalanmış, ancak bu meydana gelmeden

önce, sıfıra bölme hatasını fırlatacak bir istisna oluşmuştu. Yapılabilecek iki şey vardı. İlk aklıma gelen ikinci bir catch bloğunu dahil etmekti. Aynen aşağıda olduğu gibi,

```
try
{
   int a=1;
   int b=a/0;

   System.out.println("Dizinin 10ncu elemani "+dizi[10]);
   System.out.println("Program sonu");
}
catch(ArithmeticException e)
{
   System.out.println("Sifira bolme hatasi :");
}
catch(ArrayIndexOutOfBoundsException e)
{
   System.out.println("Indeks tasma hatasi :");
}
```



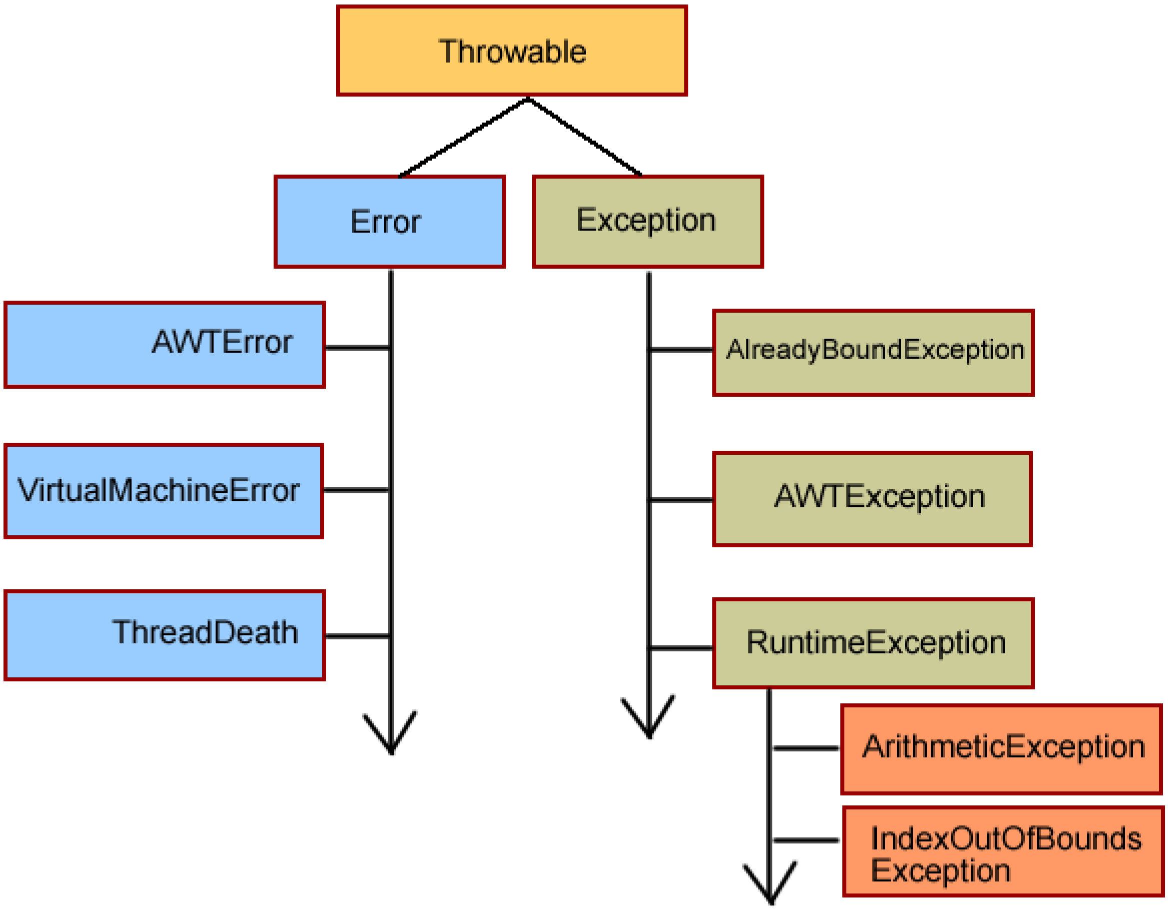
Böylece aslında birden fazla catch bloğununda nasıl kullanılabileceğini görmüş olmuştum. Diğer yandan ikinci bir yol daha vardı. Daha genel bir yol. Istisna yönetiminde, catch bloklarında sınıf nesneleri tanımlanmaktaydı. Bununla birlikte bu sınıflar Exception ismi ile bitiyorlardı. Kendimi bir anda dedektif gibi hissettim desem yeridir. Bir sonuca varmak istiyorum ama ne? Yardımıma geniş bir java dökümanı yetişti. Burada gördüğüm kadarı ile, istisna sınıfları aslında belli bir hiyerarşik düzen içerisinde yer alıyor ve birbirlerinden türetilen bir sistematik oluşturuyorlardı. Bu yapıyı zaten C# dilinden biliyordum. Ancak java�da hiyerarşi biraz daha farklıydı. Ancak ilk dikkatimi çeken, uygulamanın çalışması sırasında

yakalanabilecek düzeyde olan istisnaların Exception sınıfından türüyor olmalarıydı. Dolayısıyla, yukarıdaki örneği tek bir catch bloğu ile aşağıdaki gibi oluşturulabilirdim.

```
try
{
   int a=1;
   int b=a/0;

   System.out.println("Dizinin 10ncu elemani "+dizi[10]);
   System.out.println("Program sonu");
}
catch(Exception e)
{
   System.out.println("Hata :" +e);
}
```

Böylece, aslında try bloğunda, Exception sınıfından türeyen tüm istisna sınıfları tarafından temsil edilebilecek düzeyde oluşacak hataları yakalamış oldum. Elbette böyle bir durumda, oluşacak hataya özel kod geliştirmemiz biraz daha zor olucaktır. Çünkü hatayı en genel seviyede yakaladık. Aslında yeri gelimişken, java dilindeki istisna yönetimi sistemindende bahsetmek gerekli. Bu sistem aşağıdaki gibi bir hiyerarşiye sahip.



Elbette hiyerarşi bu kadarla sınırlı değil. Yüzlerce sınıf var. En önemli nokta, istisnaların iki ana kısımda ele alınması ve Throwable sınıfından türemiş olmaları. Throwable sınıfı, java uygulamalarının çalışması sonucu oluşabilecek hataları iki ana alt sınıfta kapsayarak değelendirmekte. Bu sınıflardan bir tanesi, try catch blokları ile yakalanamıyacak tipte olan ciddi sistem hatalarını bünyesinde barındırdan Error ata sınıfı. Diğeri ise, şu ana kadarki asıl örneklerde de incelediğim, uygulamanın çalışması sırasında yada derlenmesi sırasında , try catch tekniği ile yakalanabilecek hataları içeren Exception ata sınıfı. Örneğin benim örnek uygulamalarda incelediğim istisnalar, Exception sınıfından türeyen RuntimeException sınıfından türemiş istisnalardı. Buradaki sınıfların detaylı şekilde bilinmesi elbette mümkün değil. Ancak uygulamalarımızı geliştirirken çok iyi test etmeli ve ortaya çıkan istisna mesajlarını iyi değerlendirerek ele almalıyız diye düşünüyorum.

Gelelim istisna işleme ile ilgili diğer ilginç noktaya. C# dilinde, try blokları içerisinde, metod çağırımları kullanıldığında, bu metod çağırımlarında meydana gelebilecek hatalar ilgili catch

bloklarınca yakalanbilmekteydi. Bakalım aynı durum java dili içinde geçerli mi? Bu kez hataya neden olucak kodları bir metod içinde aldım ve try bloğu içinden çağırdım.

```
public class Istisnalar
{
    public static void main(String[] args)
    {
        int dizi[]=new int[10];

        for(int i=0;i<10;i++)
        {
            dizi[i]=i*i;
        }

        try
        {
            Metod();

            System.out.println("Dizinin 10ncu elemani "+dizi[10]);
            System.out.println("Program sonu");
        }
        catch(Exception e)
        {
            System.out.println("Hata : "+ e);
        }
    }

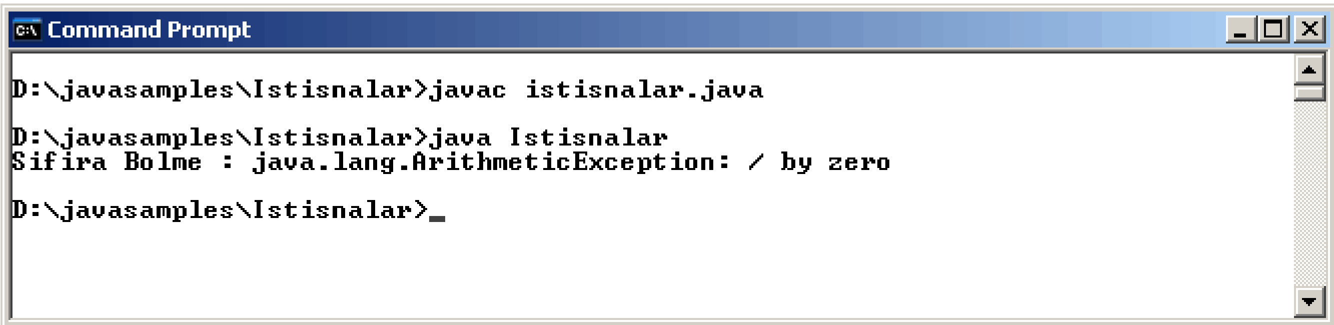
    public static void Metod()
    {
        int a=1;
        int b=a/0;
    }
}
```

Sonuç olarak uygulama başarılı bir şekilde derlendi ve çalıştı. Demekki, hatayı oluşturacak olan kodlar başka metodlar içerisinde bulunsa dahi, dolayısıyla uygulamanın başka bir konumunda olsa dahi, try bloklarınca ele alınabilir ve fırlatacakları istisnalar yakalanabilir. İstisna işlenmesi ile ilgili bir diğer ilginç konuda, iç içe geçmiş istisna bloklarının

kullanımıdır. İç içe geçmiş try blokları çoğunlukla, hataların oluştukları yerlere göre farklı davranışlarda bulunabilirler. Neden iç içe geçmiş istisna bloklarını kullanırız pek fazla fikrim yok. Ancak dilin zenginliğini bilmek açısından da oldukça önemli. Bu amaçla önce aşağıdaki gibi bir örnek geliştirdim.

```
try
{
   Metod();

   try
   {
      System.out.println("Dizinin 10ncu elemani "+dizi[10]);
      System.out.println("Program sonu");
   }
   catch(IndexOutOfBoundsException e)
   {
      System.out.println("Dizi tasmasi : "+ e);
   }
}
catch(ArithmeticException e)
{
   System.out.println("Sifira Bolme : "+ e);
}
```



Dikkatlice baktığımda gördüm ki, metodun içinde meydana gelen sıfıra bölme hatası, içteki try bloğu çalışmadan hemen önce meydana geldiğinde, içteki try blokları çalıştırılmadan, dıştaki catch bloğu çalıştırılmış ve program sonlandırılmıştı. O halde, bende try blokları gibi değişik senaryoları denemeye karar verdim. Bu kez, Metod çağırımını içteki try-catch bloğundan sonraya aldım.

```
try
{
   try
   {
      System.out.println("Dizinin 10ncu elemani "+dizi[10]);
      System.out.println("Program sonu");
   }
   catch(IndexOutOfBoundsException e)
   {
      System.out.println("Dizi tasmasi : "+ e);
   }

   Metod();
}
catch(ArithmeticException e)
{
   System.out.println("Sifira Bolme : "+ e);
}
```

```
Command Prompt

D:\javasamples\Istisnalar>javac istisnalar.java

D:\javasamples\Istisnalar>java Istisnalar
Dizi tasmasi : java.lang.ArrayIndexOutOfBoundsException: 10
Sifira Bolme : java.lang.ArithmeticException: / by zero

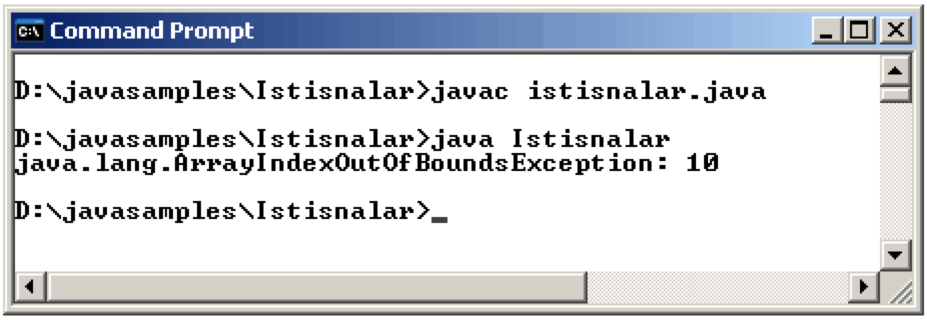
D:\javasamples\Istisnalar>
```

Ne kadar ilginç. İçteki try bloğunda meydana gelen hata içteki catch bloğunda yakalandı ama uygulama sonlanmadan diğer kod satırı devereye girdi ve dolayısıyla Metod isimli metodum çalıştırıldı. Burada oluşan hatada dıştaki catch bloğunu ilgilendirdiğinden, dıştaki catch bloğuda devreye girdi ve her iki hatada yakalandıktan sonra uygulama sonlandı. Derken aklıma şöyle bir senaryo daha geldi. İçteki try bloğunda yakalanamıyacak cinsten bir hata oluşsa ve bu hatayı yakalayacak catch dıştaki olsa, bu hata yakalanır mıydı? Bir başka deyişle acaba, içteki catch bloğunca yakalanamıyacak hata, dıştaki catch bloğunda değerlendirilmeye alınır mıydı yoksa program sonlanır mıydı? Büyük bir merak içinde hemen aşağıdaki kodları yazdım. Aklıma gelen ilk şey istisna sınıf

nesnelerinin yerlerini değiştirmek oldu. Ne kadarda yaratıcı değil mi :)

```
try
{
   try
   {
      System.out.println("Dizinin 10ncu elemani "+dizi[10]);
      System.out.println("Program sonu");
   }
   catch(ArithmeticException e)
   {
      System.out.println(e);
   }

   Metod();
}
catch(IndexOutOfBoundsException e)
{
   System.out.println(e);
}
```

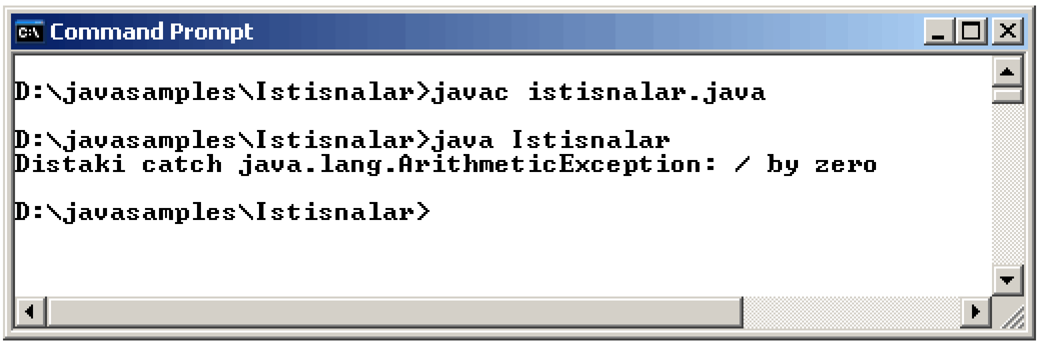


Evet şimdi şunu bir incelemekte fayda var. İçteki try bloğunda meydana gelicek olan hatayı, içteki catch bloğunun yakalayamıyacağı kesindi. Bu durumda java çalışma zamanında, dıştaki catch bloğuna geçerek, içteki try bloğunda oluşan hatayı burada aramaya karar verdi. Buradada buldu. Dolayısıyla dıştaki catch bloğunu çalıştırarak, uygulamayı bu noktada sonlandırdı. Yani, içteki try bloğunda oluşan hata dıştaki catch bloğu tarafından yakalandı.

İç içe geçmiş try-catch bloklarında oluşabilecek diğer bir durum ise, herhalde catch bloklarında oluşabilecek hataların nasıl

değerlendirilebileceği olabilirdi. Öyleki içteki catch bloğunda bir hata meydana geldiğinde bu hatanın dıştaki catch bloğu tarafından yakalanması gerekirdi. Deneyip görmek lazımdı. Çalışmak lazımdı. Durumu en iyi şekilde değerlendirebilmek için, her iki catch bloğunada aynı öncelikli bir istisna sınıfı nesnesi eklemem gerektiğini düşündüm. Bu nedenlede Exception sınıfını kullandım.

```
try
{
    try
    {
        System.out.println("Dizinin 10ncu elemani "+dizi[10]);
        System.out.println("Program sonu");
    }
    catch(Exception e)
    {
        int a=1;
        int b=a/0;
        System.out.println("Icteki catch "+e);
    }
}
catch(Exception e)
{
    System.out.println("Distaki catch "+e);
}
```

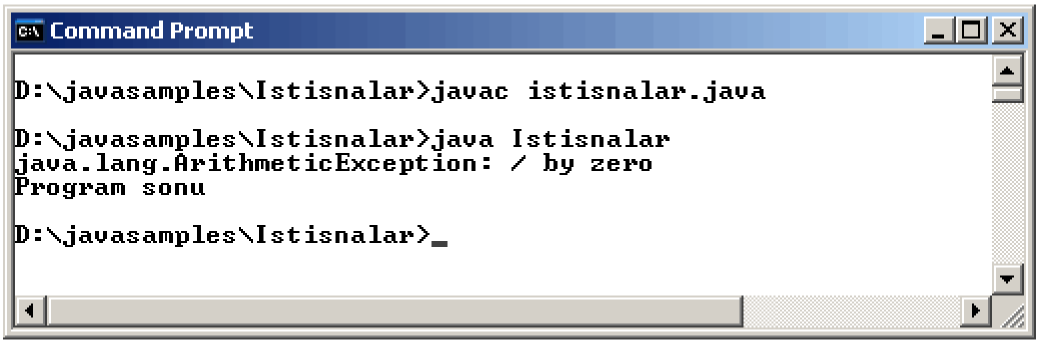


Sonuç beklediğim gibi olmuştu. İçteki catch bloğunda oluşan hata dıştaki catch bloğunca değerlendirilmeye alınmıştı. Aslında try-catch blokları ile ilgili şu ana kadar kullanmadığım bir anahtar kelime var şimdi aklıma geldi. O da finally anahtar

kelimesi. finally blokları bir try-catch kurgusunda, hata olsada olmasada devreye giren yapılardır. Örneğin;

```
try
{
   Metod();
   System.out.println("Dizinin 10ncu elemani "+dizi[10]);
}
catch(Exception e)
{
   System.out.println(e);
}
finally
{
   System.out.println("Program sonu");
}
```

Burada try bloğu içinde oluşacak olan hatanın catch bloğunda yakalanacağı kesindir. Ancak program catch bloğundan sonra hemen sonlandırılmaz. Bu noktadan sonra finally bloğu devreye girer ve içerdiği kodlar çalıştırılır. Dolayısıyla uygulamanın çıktısı aşağıdaki gibi olucaktır.



İstisna yakalama ile ilgili söylenebilecek yada düşünülebilecek şeyler sadece bunlarla sınırlı değil. Örneğin, beğenmediğimiz istisna sınıfları yerine kendi istisna sınıflarımızı yazabiliriz. Son derece kolay olan bu işlemde, tek yapmamız gereken, istisna sınıfımızı Exception sınıfından türetmemizdir. Derken tabi aklıma bir soru geldi. Her programcının bir programlama dilini öğrenirken hemen hemen her konuda sorduğu bir soru. Neden bu kadar geniş bir istisna sınıf hiyerarşisi varken kendi istisnalarımı yazmaya çalışayım ? Bunun tek nedeni bazen meydana gelen istisnanın her şeyi tam olarak açıklayamaması

olabilirdi. Buna en güzel örnek SqlException istisna sınıfıdır sanırım. Sql ile ilgili veri tabanı işlemlerinde oluşabilecek sorgu hatalarından tutunda, bağlantıda oluşabilecek hatalara kadar pek çok hatayı temsil eder. Ancak bize hata ile ilgili detaylı yada spesifik bilgiler sunmaz. Böyle bir durumda kendi istisna sınıfımızı yazma ihtiyacı duyabiliriz. Bu konu aslında geniş bir konu ve sanırım veri tabanlarının java dilinde kullanılması ile ilgili kahve molalarında inceleyeceğim.

Bölüm 14: 7 Yıl Sonra Applet

Yıllar önce ben üniversitede öğrenciyken (sanırım 1996 veya 1997 yılıydı) değerli bir sınıf arkadaşımı ziyarete gitmiştim. Kendisi bilgisayar teknolojilerine son derece ilgili birisiydi ve bu anlamda ortak pek çok yönümüz vardı. O yıllarda ikimizde, özellikle görsel programlamaya yönelik yazılım geliştirme ortamlarına ilgi duyuyorduk. O günkü ziyaretimde, dostumun elinde o güne kadar gördüğüm en kalın kitap duruyordu. Sanırım o zamanlar gözüme çok büyük gözükmüştü. Öyleki o güne dek hiç 900 sayfalık bir bilgisayar kitabı görmemiştim. Oysaki şimdi o 900 sayfalık kitapları arar oldum. En son çalıştığım bilgisayar kitabı 1500 sayfaya yakın olunca, insan ister istemez özlüyor.

Neyse sözün kısası, arkadaşımın elinde tuttuğu kitap, ingilizce bir bilgisayar kitabıydı ve Java diye bir şeyden bahsediyordu. Oha falan oldum yani der gibi arkadaşımın gözlerine baktım. Çünkü ilk aklıma gelen StarWars serisindeki Java olmuştu. Hemen ne demek istediğimi anladı ve anlatmaya başladı. Java'nın yeni bir programlama dili olduğunu, C++'ın syntax'ına çok benzer yer yer aynı yazımları kullandığını ancak işin içinde platform bağımsızlığın yer aldığını söyledi. O zamanlar bende pek çok kişi gibi platform bağımsız kısmına geldiğinde, hafif bir tebessümle hadi canım demiştim. Çok geçmeden bana kitabın ilk kaynak uygulamsından geliştirdiği kodu gösterdi. Burada komik bir çizgi karakter (kırmızı burunlu) bir internet explorer penceresinde bir oraya bir oraya taklalar atıyordu. Bu nedir diye sorduğumda bana bunun bir Applet olduğunu ve browser'ın

üzerinde dinamik olarak yerel makinede çalıştığını söyledi. O zamanlar elbetteki browser üzerinde çalışan dinamik uygulamalara hiç aşina değildim.

Java dilini öğrenmeye başladığımda, günün birinde bu değerli arkadaşımı hatırlayacağımı ve kulaklarını çınlatacağımı biliyordum. Artık o zamanlar söyledikleri şimdi kulağıma daha teknik olarak geliyor. Eeee ne demişler "geç olsunda güç olmasın". İşe appletlerin ne olduğunu kavramak ile başlamam gerekiyordu. Daha sonraki kahve molalarımda ise appletleri kullanıcı ile dinamik olarak etkileşime sokmaya çalışacaktım. Ama önce teknik bilgi ve basit kodlara ihtiyacım vardı. Tabiki appletin basit bir tanımından sonra.

Bir applet, istemci uygulamada yada başka bir deyişle yerel makinede, Java Virtual Machine'e sahip herhangibir tarayıcıda (browser) derlenerek çalıştırılan dinamik bir java programcığından başka bir şey değildir. Applet'leri normal java programları yazar gibi java dosyaları olarak yazar ve javac aracı ile class olarak byte-code'a çeviririz. Tek fark, bu program parçalarının, tarayıcıdan talep edilmeleri halinde, tarayıcının sahip olduğu JVM sayesinde derlenerek bu tarayıcının yer aldığı yerel makinede dinamik olarak çalışacak olmalarıdır. Dolayısıyla normal java byte kodları gibi, bu kodlarda çalıştırıldıklarında derlenirler. Ancak çalışma sistemleri, içerdikleri olay yapıları konsol veya görsel arabirime sahip java uygulamalarından biraz daha farklıdır. Herşeyden önce, tarayıcıda çalıştıkları için, belirli bir alan içerisinde çizilebilirler yada kullanılabilirler. Bununla birlikte dinamik çalışmaya müsait oldukları için aşağıdaki olayları gerçekleştirmelerine, yerel makinelerin güvenliği açısından izin verilmez.

Applet'lere Özgü Kısıtlamalar
Yerel makineden (çalıştıkları makine) dosya kopyalayamazlar.
Dosya silemezler.
Dosya açamazlar veya oluşturamazlar.
İndirildikleri sunucudan başka bir

sunucu ile herhangibir ağ bağlantısı kuramazlar.
İndirildikleri bilgisyarda başka programları çalıştıramazlar.
Dosya sistemine erişemezler veya okuyamazlar.

Applet'lerin çalışması ile ilgili olarak en dikkat çekici nokta, çağırıldıkları sunucudan istemci bilgisayarın tarayıcısına indirilmeleridir. Nitekim, bu işlemin gerçekleştirilmesi için, applet'e ait class dosyasının bir şekilde html kodu içerisine gömülmesi gerekecektir. Bunun nasıl yapıldığını görmek için öncelikle bir applet geliştirmek gerektiği kanısındayım. Ne kadar basit olursa olsun en azından nasıl çalıştığını görmem gerekiyor. Kaynaklarımı inceledikten sonra, aşağıdaki gibi bir örnek java dosyasını oluşturdum.

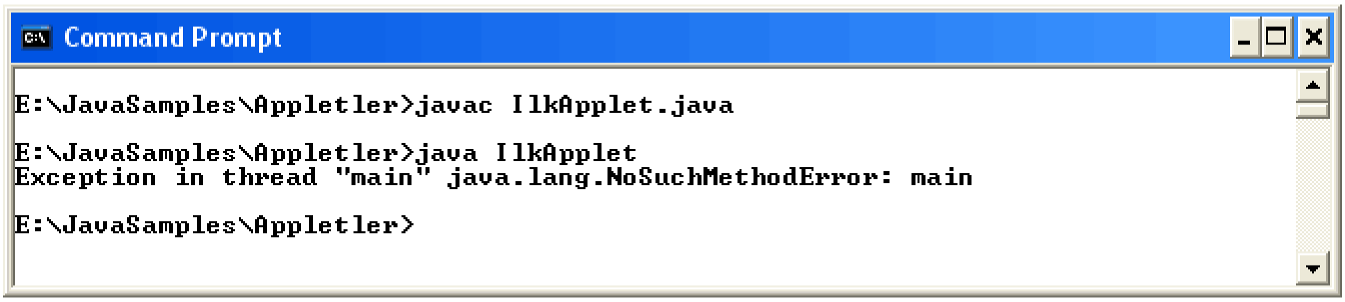
```
import java.awt.*;
import java.applet.Applet;

public class IlkApplet extends Applet
{
    public void Paint(Graphics g)
    {
            g.drawString("Yihuuu",50,50);
   }
}
```

Burada oluşturduğum java dosyasını javac ile derlediğimde herhangibir sorun ile karşılaşmadım. Peki ama kodum ne yapıyordu? Herşeyden önce ilk dikkatimi çeken, kullanılmak üzere eklediğim awt ve applet paketleriydi. Awt paketini ileride detaylı incelemeyi düşünüyordum zaten. Ancak yinede ön bilgiye ihtiyacım vardı. Awt paketi içerisinde, java ile kullanabileceğimiz görsel arayüzlere ait nesneler için bir çok sınıf bulunuyordu. Applet'lerde sonuç itibariyle, tarayıcı penceresinde çalışacaklarından, kullanıcılar ile görsel iletişim sağlamamıza yarayacak buton,textbox gibi nesneler içerebilirdi. İşte bu amaçla awt paketi vardı. Gerçi kullandığımız bir nesne

yok gibi gözükebilir ancak, Graphics sınıfı awt paketi içerisinde yer alan ve appletin çalıştığı alan içerisine bir şeyler çizmek için (örnekte olduğu gibi yazı yazmak için mesela) kullanılan bir sınıftır.

Diğer önemli bir kavramda, sınıfın Applet sınıfından türetilmiş olmasıydı. Bu, yazılan java sınıfının bir applet olarak değerlendirileceğini belirtmekteydi. Dolayısıyla applet sınıfından bir takım özellikleri kalıtımsal olarak alacağımız kesindi. Gelelim, Paint metoduna. İşte işin en can alıcı noktası burasıydı. Bu metod, tarayıcı pencersinde, appletin çalıştığı alana birşeyler çizmek için kullanılıyordu. Daha doğrusu applet, sınırları ile birlikte tarayıcı penceresinde çizilmeye başladığında çalışıyordu. Artık, değerli dostumun tarihi java kitabındaki kırmızı burunlu kahramanın nasıl taklalar attığını daha iyi anlamaya başlamıştım. O zamanlar çizgi filim gibi gelmişti. Ancak şimdi gerçeğin ta kendisi karşımdaydı. Peki şimdi ne olacak? Bir şekilde yazdığım appleti test etmem gerekiyor. İlk aklıma gelen ancak denemek istemediğim şeyi deneyerek işe başladım. Şöyleki,



Böyle birşeyin başıma geleceği kesindi diyebilirim. Elbetteki appletin çalışma sistemine bakıldığında farklı şekilde uygulanmaları gerekirdi. Her şeyden önce, bu applet bir tarayıcıya indirilecek ve oradaki JVM tarafından derlenecekti. Bunu test etmenin iki yolu vardı. Birincisi bir applet tagı ile bu sınıfı bir html sayfasına koymak yada Applet Viewer aracını kullanmaktı. İlk önce applet tagını aşağıdaki gibi denedim. Bunun için applet sınıfım ile aynı klasörde olan bir html sayfası hazırladım.

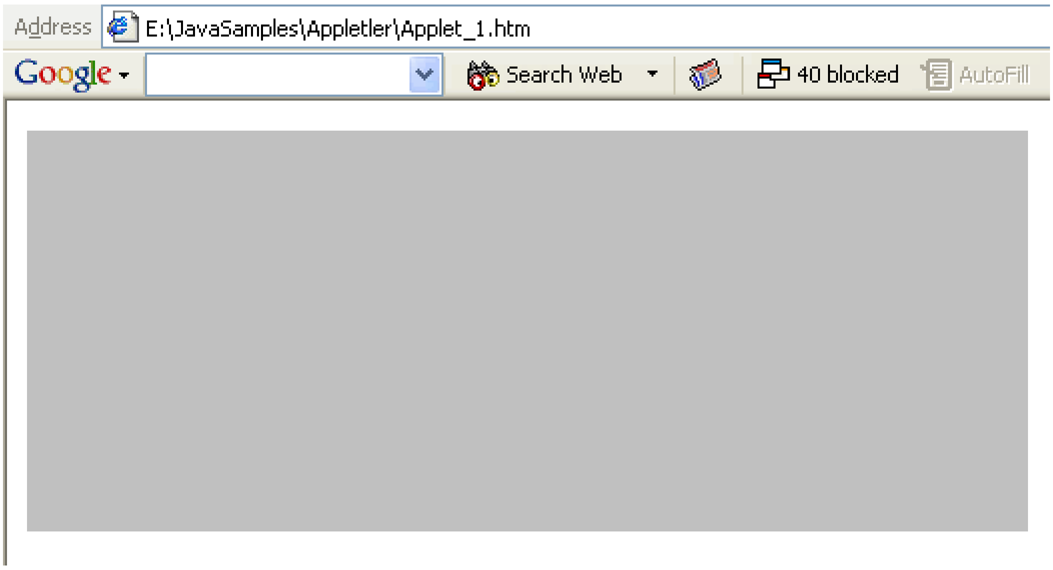
```
<html>
```

```
<head>
<meta http-equiv="Content-Language" content="tr">
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<title>New Page 1</title>
</head>

<body>
<APPLET CODE="IlkApplet.class" WIDTH="500"
HEIGHT="200">
</APPLET>
</body>

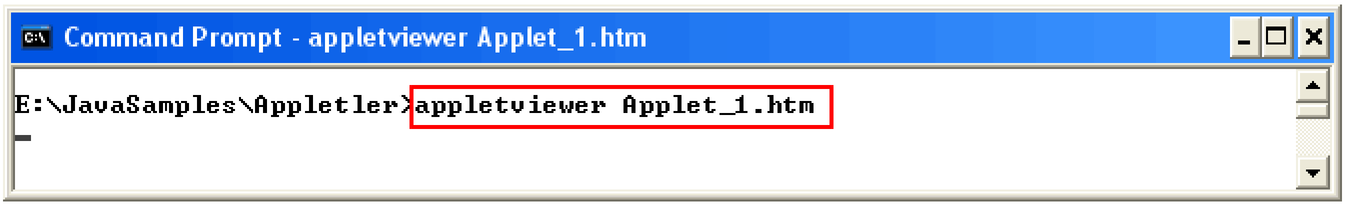
</html>
```

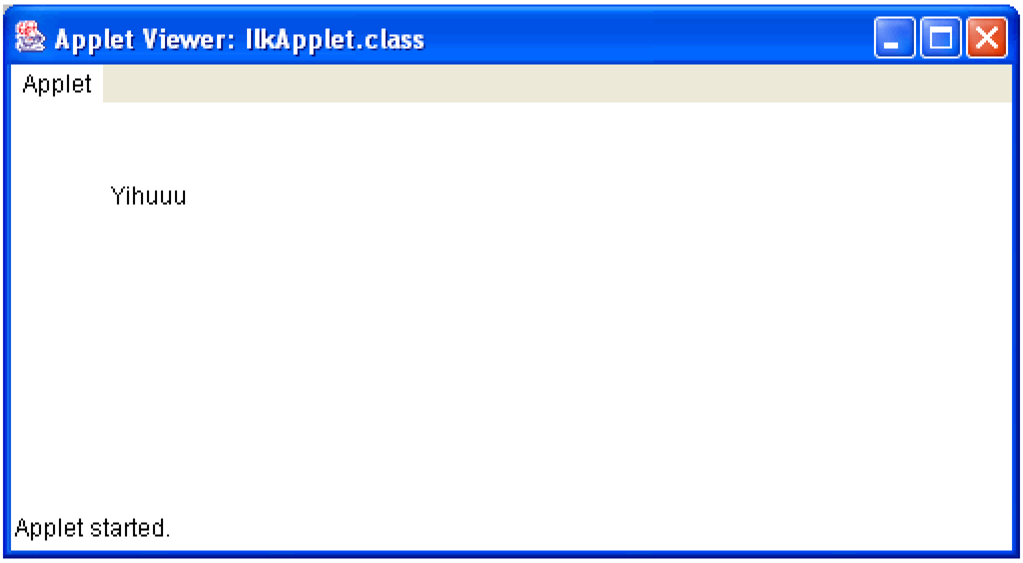
Applet tagı içinde en önemli kısım CODE anahtar kelimesinin olduğu kısım idi. Burada, belirtilen 500 piksel genişlik ve 200 piksel yüksekliğindeki alanda hangi applet sınıfının çalıştırılacağını belirtiyorduk. Şimdi oluşturduğum bu html sayfasını tarayıcıda açtım. Ancak hiç beklemediğim aşağıdaki sonucu elde ettim.



500 piksel'e 200 piksellik bir alan açılmıştı. Ancak yazmak istediğim yazıyı görememiştim. Bunun tek nedeni olabilirdi. JVM, ya sınıf dosyasını derlememişti yada appet sınıfını tarayıcı penceresine indirememiştim. Tabi bir diğer ihtimalde tarayıcının özelliklede Microsoft Internet Explorer olduğu için, JVM

desteğinin kaldırılmış olabileceğiydi. Aklıma ilk gelen en güncel java plug-in indirmek oldu. Ancak daha öncesinde en azından yazdığım appletin doğru olup olmadığından emin olmalıydım. Neyseki, java'nın appletviewer aracı yardımıma yetişti. Komut satırında aşağıdaki satır ile appletimin çalışmasının sonucunu gördüm. Applet Viewer programı, yazılmış olan appletlerin tarayıcı penceresine ihtiyaç duyulmadan çalıştırılabilmelerini sağlıyordu.



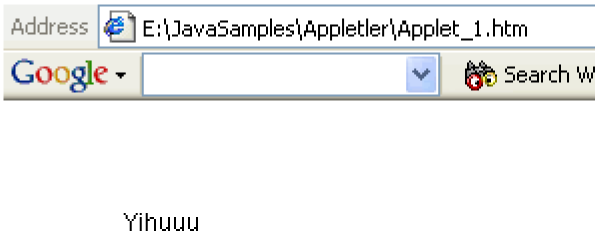


Evet appletim çalışmıştı. Applet viewer bir applet'i test etmek için ideal bir yoldu. Ama kafam halen daha internet explorer tarayıcısında neden çalışmadığındaydı. Hemen internete girdim ve java plug-in için en güncel sürümü aradım.

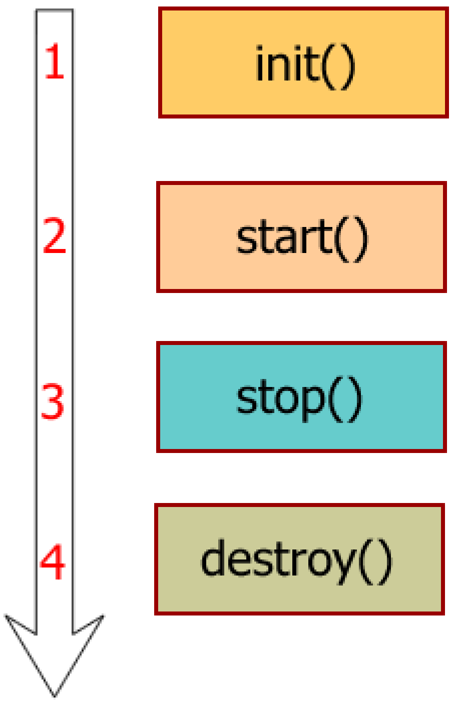
http://java.sun.com/products/plugin/reference/codesamples/index.html

Bu adreste örnek java appletleri vardı. En son sürüme ait olanlardan bir tanesini çalıştırmak istediğimde, JVM için gerekli sürümü yüklemek isteyip istemediğimi sordu. Tabiki bunu istiyordum. Hemen yükledim. Hemen derken yüklemek biraz zaman aldı tabiki ama sonuça herşey yoluna girdi. Bu işlemin

sonucunda html sayfamı tarayıcıdan tekrar çalıştırdığımda aşağıdaki sonucu elde ettim.



Appletim html sayfasından da çalışmıştı. Harika. Bu appletin ardından daha gelişmiş bir applet yazmam gerektiğini düşünüyordum ki karşıma appletlerin çalıştırıldığında gerçekleşen olayların bir listesi geliverdi. Bir applet çalıştırıldığında aslında aşağıdaki olaylar gerçekleştiriliyordu.



Görüldüğü gibi bir appletin çalışması sırasında işleyen 4 temel olay var. Bu metodlardan ilki init metodu, applet tarayıcı bilgisayara indirildiğinde çalıştırılmaktadır. Start metodundaki kod satırları ise applet çalışmaya başladığında tetiklenir. Stop metodunda yer alan kodlar, appletin bulunduğu sayfadan başka bir sayfaya atlandığında dolayısıyla applet kapatıldığında çalıştırılır. Destroy metodundaki kodlar ise, tarayıcı penceresi kapatıldığı sırada çalıştırılır. Elbette birde paint metodumuz var. Bu metod ile, appletin içerisinde tarayıcı penceresinde

belirlenen alanlarda birşeyler çizdirmek için kullanacağımız kodlar yer alır. Diğer yandan, kullanıcı ile etkişim halinde olan appletlerde, kullanıcının tepkisine göre applet üzerinde yapılacak yeni çizimler repaint isimli metodlar içerisinde gerçekleştirilir.

Şimdi bana bu metodların bir appletin çalışması sırasında nerelerde devreye girdiğini gösterecek bir örnek gerekliydi. Hemde appleti biraz daha geliştirmiş olurdum. Bu amaçla kaynaklarımdan edindiğim bilgiler ışığında aşağıdaki gibi bir java applet sınıfı oluşturdum.

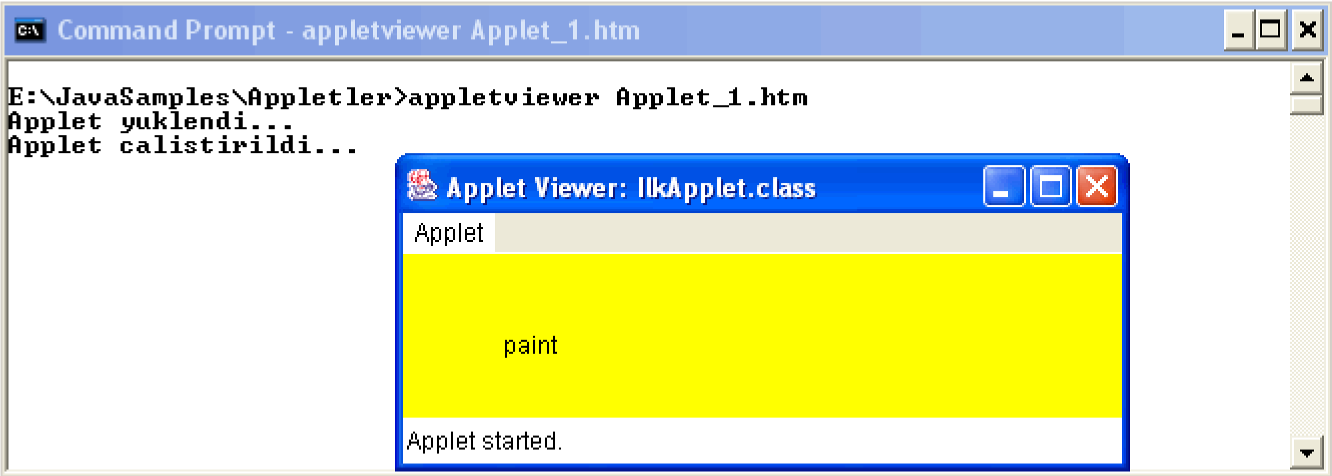
```
import java.awt.*;
import java.applet.Applet;

public class IlkApplet extends Applet
{
   public void init()
   {
      setBackground(Color.yellow);
      System.out.println("Applet yuklendi...");
   }

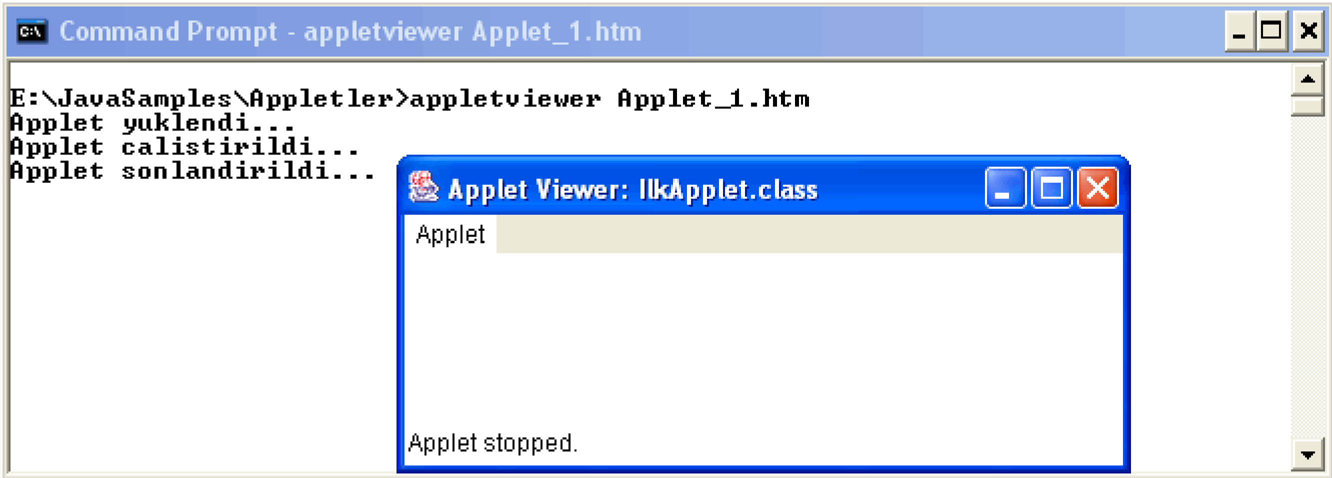
   public void paint(Graphics g)
   {
      g.drawString("paint",50,50);
   }

   public void start()
   {
      System.out.println("Applet calistirildi...");
   }
   public void stop()
   {
      System.out.println("Applet sonlandirildi...");
   }
}
```

Bu applet sınıfını derleyip appletviewer ile çalıştırdığımda ilk olarak aşağıdaki görüntüyü elde ettim.



Görüldüğü gibi ilk önce init metodu devreye girdi. Ardından applet'in start metodunda yer alan kodlar çalıştırıldı ve sonrasında ise paint metodundaki kodlar devreye girdi. Çalışan bu applet'i Appet Viewer'ın Applet menüsünden stop komutu ile durdurduğumda ise aşağıdaki ekran görüntüsünü elde ettim.



Bu kez applet'in stop metodundaki kodlar devreye girmişti ve applet'in çalışmasıda sona ermişti. Applet'lerin çalışma sistemini anladıktan, yaşam süresi boyunca çalıştıracağı metodları ve gerçekleşen olayları inceledikten ve bir buçuk satırlık applet kodu yazdıktan sonra, daha işe yarar bir örnek görmek istiyordum. Hatta yazmak istiyordum. Ancak işe yaramasa bile beni etkileyebilecek bir örnek bulmanın daha iyi olacağı kanısına vardım. Bu amaçla Sun'ın sitesinden örnek appletlere baktım.

http://java.sun.com/products/plugin/1.5.0/demos/plugin/applets/MoleculeViewer/example2.html

Bu adreste yer alan applet'i güzelce bir inceledim. Applet'te yapılması gereken, mouse'a basılı tutup şekli herhangibir yöne doğru sürüklemeye çalışmaktı. Açıkçası bu applet'e bakınca ve şu an java dilinde bulunduğum yeri düşününce kendi kendime şöyle dedim. "ÇOOOOK ÇALIŞMAM LAZIMMM. ÇOOOOKK!!!". Neyseki önümüzdeki hafta boyunca, java appletlerinde awt sınıfına ait GUI nesnelerini kullanarak kullanıcılar ile nasıl dinamik etkileşime geçileceğini öğrenmeye çalışacağım. Artık dinlenmenin tam zamanı. Kahvemde bitmiş zaten.

Bölüm 15: Appletler ile Görsel Tasarım

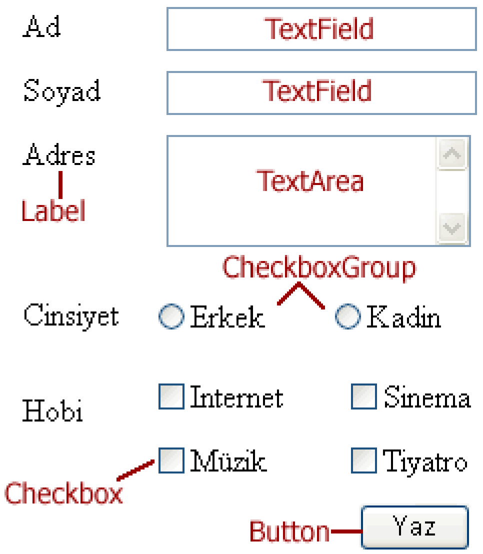
Geçen hafta boyunca Applet' lerin büyülü dünyasını daha çok keşfedebilmek için uğraştım durdum. Nitekim Applet' leri daha etkin bir şekilde kullanabileceğimi ve Applet'lerin çok daha fazlasını sunduğunu biliyordum. Örneğin, kullanıcılar ile dinamik etkileşime geçilmesini sağlayacak tarzda applet' ler üzerinde çalışabilirdim. Bu konuda aklıma ilk gelen, bilgi giriş formu ekranı oldu. İlk başta nereden başlamam gerektiğini bilmiyordum. Applet' lerin çalışma mimarisinden haberim vardı. Ancak, görsel öğeleri bu applet' ler üzerinde nasıl oluşturabilirdim? Dahada önemlisi, bu görsel nesnelerin, kullanıcı aktivitelerine cevap vermesini nasıl programlayabilirdim? Bir başka deyişle, görsel programlamanın en önemli yapıtaşlarından birisi olan olay-güdümlü programlamayı nasıl gerçekleştirebilirdim? Tüm bu soruların cevaplarını bulmak maksadıyla, hafta boyunca araştırmalarımı sürdürdüm.

Kilit nokta, Java dilinin Awt isimli (Abstract Windows Toolkit) paketiydi. Awt hem applet' ler için hemde ileride incelemeyi düşündüğüm normal GUI (Graphical User Interface) ler için ortak nesne modellerini kapsülleyen bir paketti. İlk okuduğumda bu paketin, Voltran' ın parçalarından birisi olduğunu zannetmiştim. Ancak sağladığı imkanlar ile, Voltran' ın değil gövdesi tüm benliğini oluşturabilirdim. İşin geyiği bir yana, Awt paketi, java ile geliştirilen herhangibir GUI uygulaması için gerekli olan tüm görsel bileşenleri sağlamaktaydı. Hatta bu

bileşenlerin her GUI uygulamasında aynı tipte görünmesinede imkan tanıyordu.

Özellikle Visual Studio.Net gibi görsel geliştirme ortamlarında program arayüzlerini (interface programlama değil, görsel tasarım anlamında) tasarlamak son derece kolay. Ama ister .net platformunda olsun ister Java platformunda, nesne yönelimli dillerin doğası gereği tüm görsel bileşenlerde aslında birer sınıf modelinin örneklemelerinden başka bir şey değiller. Dolayısıyla Awt paketi içindeki görsel bileşenlerinde birer sınıf modeli olduğunu belirtmekte yarar var. Örneğin, sayfalarda gösterebileceğim butonlar Button sınıfına ait nesne örnekleri olacak. Sadece metin bilgisi taşıyan okuma amaçlı Label bileşenleri, Label sınıfına ait olacak. Yada Checkbox, TextField, TextArea, List, Image kontrolleri vs...

Artık bir noktadan başlamam gerektiğini düşünüyordum. Kahvemden bir yudum aldım ve ilk önce nasıl bir form tasarlamak istediğime karar verdim. Bunu kağıt üzerinde çizmek kolaydı ancak dijital ortama aktarmak zordu. Tasviri tam yapmak için, Vs.Net editörünü kullandım ve aşağıdaki gibi bir formu, applet olarak tasarlamaya karar verdim.



Bu formu oluşturmak için hangi sınıfları kullanmam gerektiğinede, JSDK'dan baktım. Şu ana kadar her şey açık ve netti. Şimdi sıra kodlama kısmına gelmişti. Aslında ne yapmam gerektiğini açıkça tahmin edebiliyordum. Applet, bileşenleri üzerinde barındıracak yer olduğuna göre, Applet' i oluştururken,

başka bir deyişle applet' i çalışır hale getitirken bu nesneleri yükleyebilirdim. Ancak bundan önce bu bileşen nesnelerini oluşturmam ve daha sonra bir şekilde Applet'e eklemem gerekiyordu. Felsefe işte bu kadar basitti. Felsefenin asıl karışacağı noktanın, bu bileşen nesnelerinin olaylara cevap verebilecek şekilde kodlanmasında oluşacağını da hissediyordum. Ancak ilk etapta, öncelikle appleti tasarlamam ve en azından bileşenleri applet üzerinde sorunsuz bir şekilde göstermem gerektiği kanısındaydım. Hemen kolları sıvadım ve aşağıdaki applet sınıfını oluşturdum.

```
import java.awt.*;
import java.applet.Applet;

public class Gui_1 extends Applet
{
    public void init()
    {
        setBackground(Color.white);

        Label lbAd=new Label("Ad ");
        Label lbSoyad=new Label("Soyad ");
        Label lbAdres=new Label("Adres ");
        Label lbCinsiyet=new Label("Cinsiyet ");
        Label lbHobi=new Label("Hobi ");

        TextField txtAd=new TextField();
        TextField txtSoyad=new TextField();

        TextArea txtAdres=new TextArea(2,5);

        CheckboxGroup cbCinsiyet=new CheckboxGroup();

        Checkbox cbInternet=new Checkbox("Internet");
        Checkbox cbMuzik=new Checkbox("Muzik");
        Checkbox cbSinema=new Checkbox("Sinema");
        Checkbox cbTiyatro=new Checkbox("Tiyatro");

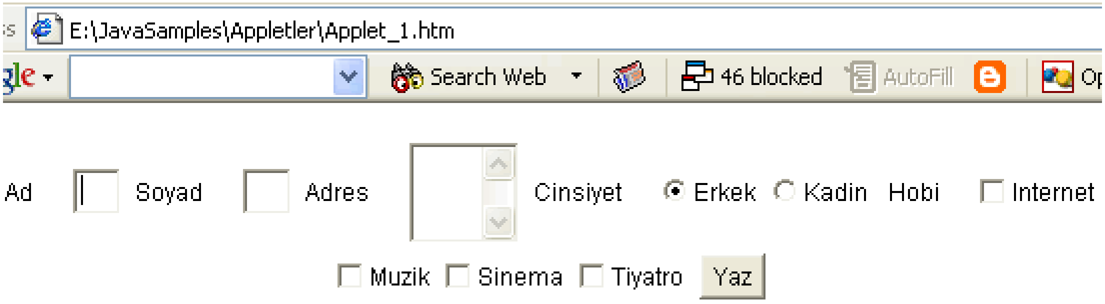
        Button btnYaz=new Button("Yaz");
```

```
        add(lbAd);
        add(txtAd);
        add(lbSoyad);
        add(txtSoyad);
        add(lbAdres);
        add(txtAdres);
        add(lbCinsiyet);
        add(new Checkbox("Erkek",cbCinsiyet,true));
        add(new Checkbox("Kadin",cbCinsiyet,false));
        add(lbHobi);
        add(cbInternet);
        add(cbMuzik);
        add(cbSinema);
        add(cbTiyatro);
        add(btnYaz);
    }
}
```

Bu uzun kod dosyasında temel olarak yapılan işlemler çok basitti. Herşeyden önce nesneleri oluşturabileceğim en uygun yer init metoduydu. Burada her bir görsel bileşen nesnesini teker teker new operatörü ile oluşturdum. Label bileşenleri için yapıcılarına Label'ın başlık metnini gönderdim. Aynı işlemi, CheckBox bileşenleri ve Button bileşeni içinde yaptım. Böylece ekrandaki Label, Checkbox ve Button bileşenlerinin başlıklarının ne olacağınıda otomatik olarak belirlemiş oldum. Textfield bileşenleri içinde parametreler girilebilir. Özellikle Textfield' ın boyutunu belirlemek için. Textarea bileşeni için ise, iki parametre girdim. İlki satır sayısını ikinciside sütun sayısının göstermekte. Burada tek özel oluşum radyo buton dediğim CheckboxGroup bileşenine ait. Bu bileşene, yine Checkbox bileşenleri ekleniyor. Yani bir CheckboxGroup bileşeni birden fazla Checkbox kontorlünü aynı isim altında gruplamak ve böylece bu bileşenlerden sadece birisinin seçili olmasını garanti etmek amacıyla kullanılmakta.

Bu bileşen örneklerinin oluşturulmasından sonra tek yaptığım add metodunu kullanarak bunları Applet' imin üzerine eklemek oldu. Sonuç mu? Aslında göstermeye çekiniyorum. Çünkü hiçte umduğum gibi değil. Java bytecode dosyasının class olarak

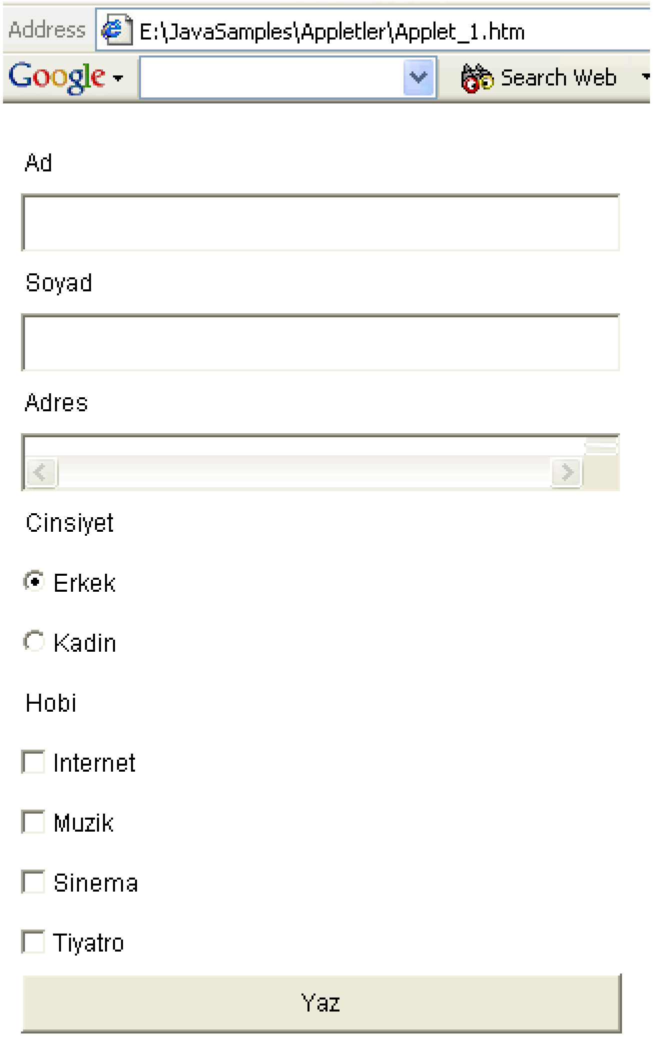
derledikten sonra bir html sayfasına <Applet tagını kullanarak ekledim. İşte sonuç;



Ahhhh!!! Ahhh. Nerede güzelim Visual Studio.Net, Dreamweaver, Frontpage....İyi güzel bileşenleri oluşturmuştum, applet' ede başarılı bir şekilde eklemiştim. Ama ya görsel tasarımın zerafeti ne olacak. Tam bu noktada işte çakılıp kalmıştım. Kaynaklarıma baktım benim yaptığım bu estetik abidesini onlarda başarmışlardı. Kaynaklarımı biraz daha araştırdıktan sonra aslında bu tasarım ve sayfaya yerleştirme işinin bazı kitaplarda bölüm olarak işlendiğini gördüm. Ancak şu an için bana en azından ızgaralanmış bir görümü hazırlayabileceğim bir teknik gerekiyordu. Sonunda buldum ama... Gerçi bulana kadar bir kaç fincan kahveyide bitirdim. İşin sırrı Olinde değil, tabiki sağdakindeydi. Yani Layout tekniği. Uyguladığım teknik GridLayout tekniği. Tek yapmam gereken tüm kontrolleri eklemeden önce, Applet üzerinde bir GridLayout yani ızgara belirlemekti. Bunu gerçekleştirmek için Applet'in init metodunun en başına aşağıdaki kod satırını ekledim.

```
setLayout(new GridLayout(15,2));
```

Bu satır ile Applet'in web sayfasında kaplayacağı alanı, 15 satır ve 2 sütuna bölmüştüm. Artık GUI bileşenleri sırasıyla yerleşecekti. Ancak yinede sonuç istediğim gibi olmamıştı.



Hiç yoktan iyidir diyerek devam etmek zorundaydım. Ancak Layout ayarlamaları ile ilgili olarak başka bir kahve molasında daha derin bir araştırma yapmayıda kafama koymuştum. Olay yerinden uzaklaşırken, en azından applet üzerinde dinamik olarak görsel bileşenlerin nasıl eklendiğini anlamış ve bir kaç bileşenide öğrenmiştim. Asıl merak ettiğim, butona basıldığında olmasını istediklerimi nasıl yazacağımdı? Bunun için, C# dilinde özellikle görsel programlamada delegeler ile yakın ilişkide olan event'lar kullanılıyordu. Java dilindede durum çok farklı değildi ancak anlaşılması daha kolaydı. Java dilindede, kullanıcı tepkilerini ele alabilmek için delegasyon mantığı kullanılıyordu. Bu modelin en önemli yanı, görsel bileşenlerinin kullanıcı tepkilerini algılayabilmelerini istediğimiz Applet sınıfına, ActionListener arayüzünü uygulamamız gerekliliğiydi. Kolları sıvadım ve ilk olarak, en basit haliyle, Button bileşenime tıklandığında meydana gelecek kodları hazırladım.

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Gui_1 extends Applet implements ActionListener
{
   TextField txtAd;
   TextField txtSoyad;
   Button btnYaz;

   public void init()
   {
      setBackground(Color.white);

      Label lbAd=new Label("Ad ");
      Label lbSoyad=new Label("Soyad ");
      Label lbAdres=new Label("Adres ");
      Label lbCinsiyet=new Label("Cinsiyet ");
      Label lbHobi=new Label("Hobi ");

      txtAd=new TextField();
      txtSoyad=new TextField();

      TextArea txtAdres=new TextArea(2,5);

      CheckboxGroup cbCinsiyet=new CheckboxGroup();

      Checkbox cbInternet=new Checkbox("Internet");
      Checkbox cbMuzik=new Checkbox("Muzik");
      Checkbox cbSinema=new Checkbox("Sinema");
      Checkbox cbTiyatro=new Checkbox("Tiyatro");

      btnYaz=new Button("Yaz");

      setLayout(new GridLayout(15,2));

      add(lbAd);
      add(txtAd);
      add(lbSoyad);
      add(txtSoyad);
```

```
      add(lbAdres);
      add(txtAdres);
      add(lbCinsiyet);
      add(new Checkbox("Erkek",cbCinsiyet,true));
      add(new Checkbox("Kadin",cbCinsiyet,false));
      add(lbHobi);
      add(cbInternet);
      add(cbMuzik);
      add(cbSinema);
      add(cbTiyatro);
      add(btnYaz);

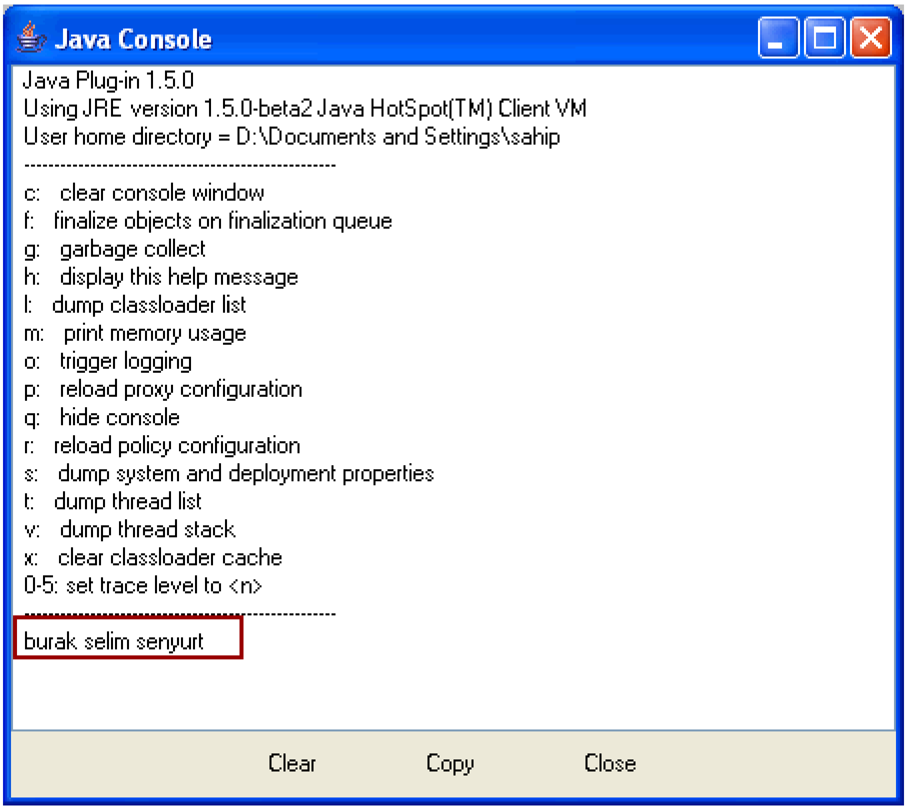
      btnYaz.addActionListener(this);
   }

   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==btnYaz)
      {
         System.out.println(txtAd.getText()+"
"+txtSoyad.getText());
      }
   }
}
```

Burada kullanılan tekniği anlamanın çok önemli olduğunu düşünüyorum. Öncelikle Applet'in üzerindeki nesnelerin olaylarını dinleyecek şekilde programlamalıydım. Bunu gerçekleştirebilmek için, sınıfa ActionListener arayüzünü uyguladım. Ardından hangi bileşen için bir olay dinlemesinin gerçekleştirilmesini istiyorsam, o nesne içi addActionListener metodunu this parametresi ile kullandım. Böylece, bu örnekteki Button bileşenine bu applet içinde yapılacak tıklamalar ele alınabilecek ve kodlanabilecekti. Başka bir deyişle, nesneye ait olası olayların dinlemeye alınmasını sağlamıştım. Bu işlemin ardından elbetteki, olay meydana geldiğinde çalıştırılacak kodları yazacağım bir metod gerekliydi. İşte buda, Button bileşenlerine yapılan tıklamaları dinleyen actionPerformed metodu. Bu metod ActionEvent isimli bir parametre alıyor. Bu parametre sayesinde, hangi buton bileşenine tıklandığını

dinleyebilirim. İşte böylece buton bileşenine basıldığında işleyecek olan satırları burada yazmış oldum. Şimdi bu sistemi deneme vakti geldi.

Tarayıcıda sayfamı çalıştırdım txtAd ve txtSoyad kontrollerine isim ve soyisim bilgilerimi girdim buton' a tıkladım ve ekranın sağ alt köşesindeki System Tray'da yer alan JVM kahve sembölünden, Open Console diyerek, console penceresine geçtim. Sonuç olarak tıklama olayım algılanmış ve olaya karşılık gelen kod satırları çalıştırılmıştı.



Şimdi aklıma takılan başka bir nokta vardı. Eğer applet'imde iki button bileşeni olsaydı. Her biri için ayrı ayrı olay dinleyicilerimi yazacaktım? Nitekim, actionPerformed metodunun yapısı buna müsait değildi. Bu amaçla, applet' e TextField ve TextArea kontrollerinin içeriğini temizleyecek yeni bir Button bileşeni ekledim. Kodun son hali aşağıdaki gibi oldu.

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
```

```
public class Gui_1 extends Applet implements ActionListener
{
   TextField txtAd;
   TextField txtSoyad;
   TextArea txtAdres;

   Button btnYaz;
   Button btnSil;

   public void init()
   {
      setBackground(Color.white);

      Label lbAd=new Label("Ad ");
      Label lbSoyad=new Label("Soyad ");
      Label lbAdres=new Label("Adres ");
      Label lbCinsiyet=new Label("Cinsiyet ");
      Label lbHobi=new Label("Hobi ");

      txtAd=new TextField();
      txtSoyad=new TextField();

      txtAdres=new TextArea(2,5);

      CheckboxGroup cbCinsiyet=new CheckboxGroup();

      Checkbox cbInternet=new Checkbox("Internet");
      Checkbox cbMuzik=new Checkbox("Muzik");
      Checkbox cbSinema=new Checkbox("Sinema");
      Checkbox cbTiyatro=new Checkbox("Tiyatro");

      btnYaz=new Button("Yaz");
      btnSil=new Button("Sil");

      setLayout(new GridLayout(16,2));

      add(lbAd);
      add(txtAd);
      add(lbSoyad);
      add(txtSoyad);
```

```
        add(lbAdres);
        add(txtAdres);
        add(lbCinsiyet);
        add(new Checkbox("Erkek",cbCinsiyet,true));
        add(new Checkbox("Kadin",cbCinsiyet,false));
        add(lbHobi);
        add(cbInternet);
        add(cbMuzik);
        add(cbSinema);
        add(cbTiyatro);
        add(btnYaz);
        add(btnSil);

        btnYaz.addActionListener(this);
        btnSil.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource()==btnYaz)
        {
            System.out.println(txtAd.getText()+"
"+txtSoyad.getText());
        }
        else if(e.getSource()==btnSil)
        {
            txtAd.setText("");
            txtSoyad.setText("");
            txtAdres.setText("");
            repaint();
        }
    }
}
```

İlk hali;



Sil başlıklı Button bileşenine tıklandıktan sonraki hali.



Her iki buton bileşenide, aynı olay dinleyicisi içerisinde ele alınmıştı. actionPerformed dinleyicisinde, hangi butona tıklandığı, ActionEvent parametresinin getSource metodu ile tespit edilmekteydi. Elbetteki awt paketinde yer alan diğer görsel bileşenler için sadece bu olay dinleyicisi söz konusu değildi. İşin aslı, nesneler üzerindeki kullanıcı etkilerine göre dinleyiciler tanımlanmıştı. Örneğin, CheckBox bileşenine tıklanması bu bileşenin durumunun değişmesi anlamına gelmekteydi ki bu durumda, ItemListener dinleyici metodu bu etkiyi ele alacaktı. Diğer nesneler içinde benzer durumlar söz konusu. Ama ne zaman? Bir dahaki kahve kokusunda.

Bölüm 16: Appletler ve JBuilder

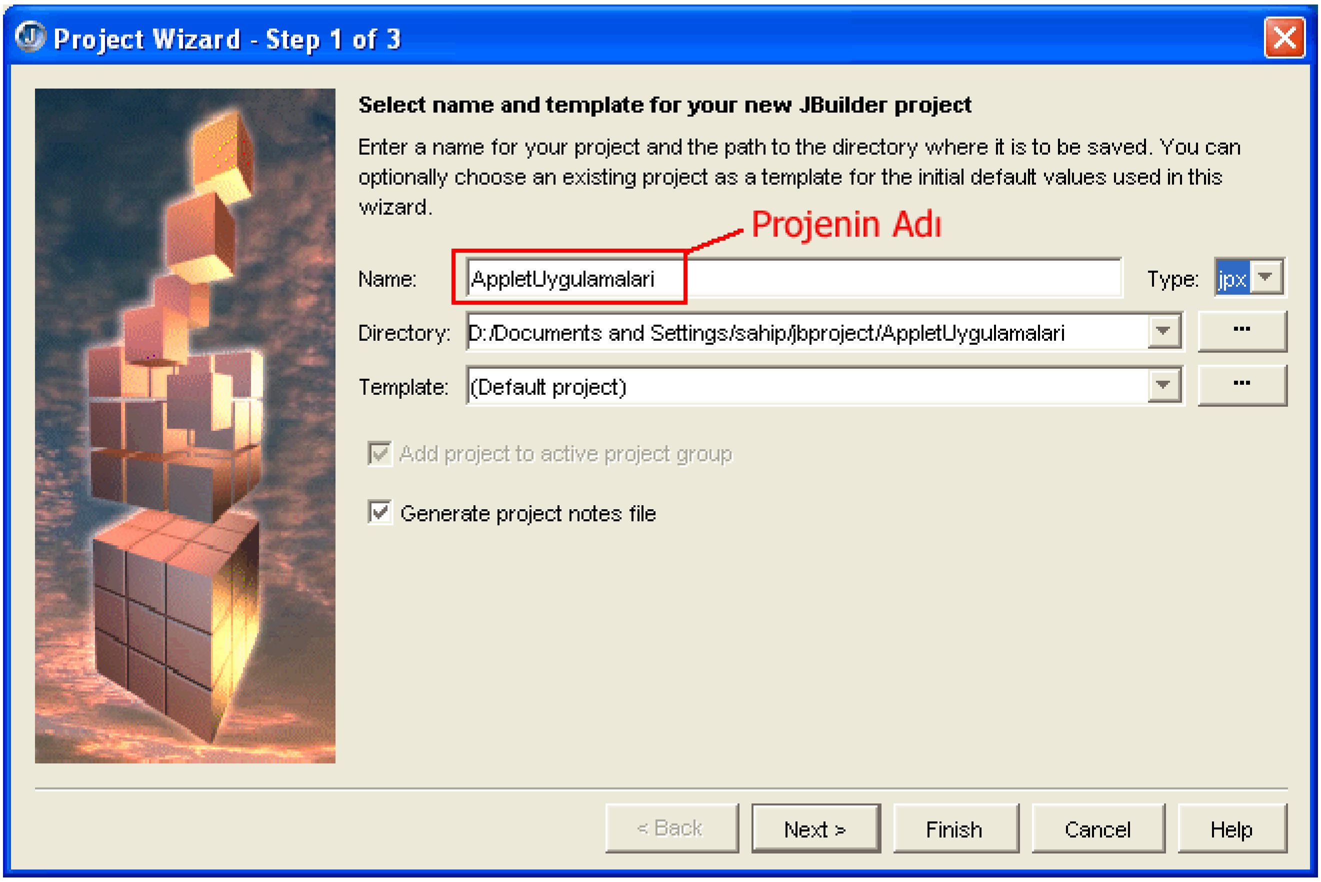
Geçen hafta boyunca, Java dili ile Applet' lerin nasıl tasarlandığını incelemeye çalıştım. Mimari kısmında yer alan hemen herşeyi incelemiştim. Bir AWT bileşenin bir Applet' e

nasıl ekleneceğini, bu bileşenin bir takım temel özelliklerinin nasıl ayarlanacağını, bu bileşene bağlı olay dinleyicilerinin ne şekilde uygulanması gerektiğini vs... Ancak karşılaştığım en uğraştırıcı sorun, bileşenlerin bir Applet üzerinde yerleşimlerinin Layout' lar vasıtasıyla ayarlanmaya çalışmasıydı. Açıkçası, VS.NET gibi bir geliştirme ortamının sunduğu görsel tasarım rahatlığını özlemiştim. Bu düşünceler içerisinde Kadıköy' ün arka sokaklarında yürürken çok sevgili bir dostum ile karşılaştım. Kendisinin sıkı bir Java programcısı olduğunu biliyordum. Beni ofisine davet etti.
Ofiste bulunduğum süre zarfında konuştuğumuz ve tartıştığımız tek konu Applet' lerin ve diğer görsel java uygulamalarının kolay bir şekilde tasarlandığı ve daha çok uygulamanın nasıl çalışacağı ve kodlamaları ile ilgilenilebildiği, bir başka deyişle asıl önemli olan konulara zaman ayrılabildiği, vs.net tarzı bir geliştirme ortamının var olup olmadığıydı. Değerli arkadaşım yüzünde kurnaz bir tebessüm ile, "Sana göstermek istediğim bir program var" dedi. Bilgisayarın başına geçtik ve ta taaa... JBuilder Enterprise 9! Adını ve methini çok duyduğum bu yazılım geliştirme ortamını daha önce hiç denememiştim. Arkadaşım, "Ben java ile geliştirdiğim uygulamaları burada yazıyorum." dedi. Gözlerimde bir anda bir ışıltı parladığını hissettim. Tek söylediğim "Yeni makalemi bugün burada yazabilir miyim?" olmuştu.

Her yazılımcı, geliştireceği bir uygulama için öncelikle, istekleri ve talepleri değerlendirerek işe başlar. Sonra sistemin analizini ve tasarlanmasını gerçekleştirir. Gerekirse UML şemaları yardımıyla yazılımın tasarı planınıda oluşturur. Sonraki adım ise, kodlamaların yapılmasıdır. Ancak çoğu zaman ne ben ne de diğer pek çok yazılımcı Notepad gibi muhteşem bir editor ile bu tarz projeleri geliştirmek istemez. Çünkü bu tarz bir yöntem izlendiğinde, yazılımın önem arz eden konularına ayrılacak olan zaman, notepad ile yazılan kodlarda, bileşenlerin ekran üzerine düzgün bir şekilde yerleştirilmesinin sağlanması, olay dinleyicilerinin ayarlanması gibi gereksiz yere zaman alıcı konulara harcanır. Eğer bu böyle olmasaydı çoğumuz bu gün C# ile geliştireceğimiz projeleri Vs.Net gibi bir ortamda yazmazdık. İşte JBuilder' da Java ile yazılım geliştirmek üzere tasarlanmış bir yazılım geliştirme aracıydı. Çok şükür ki, arkadaşımda bu yazılımın olması benim bu ortamı test etmem

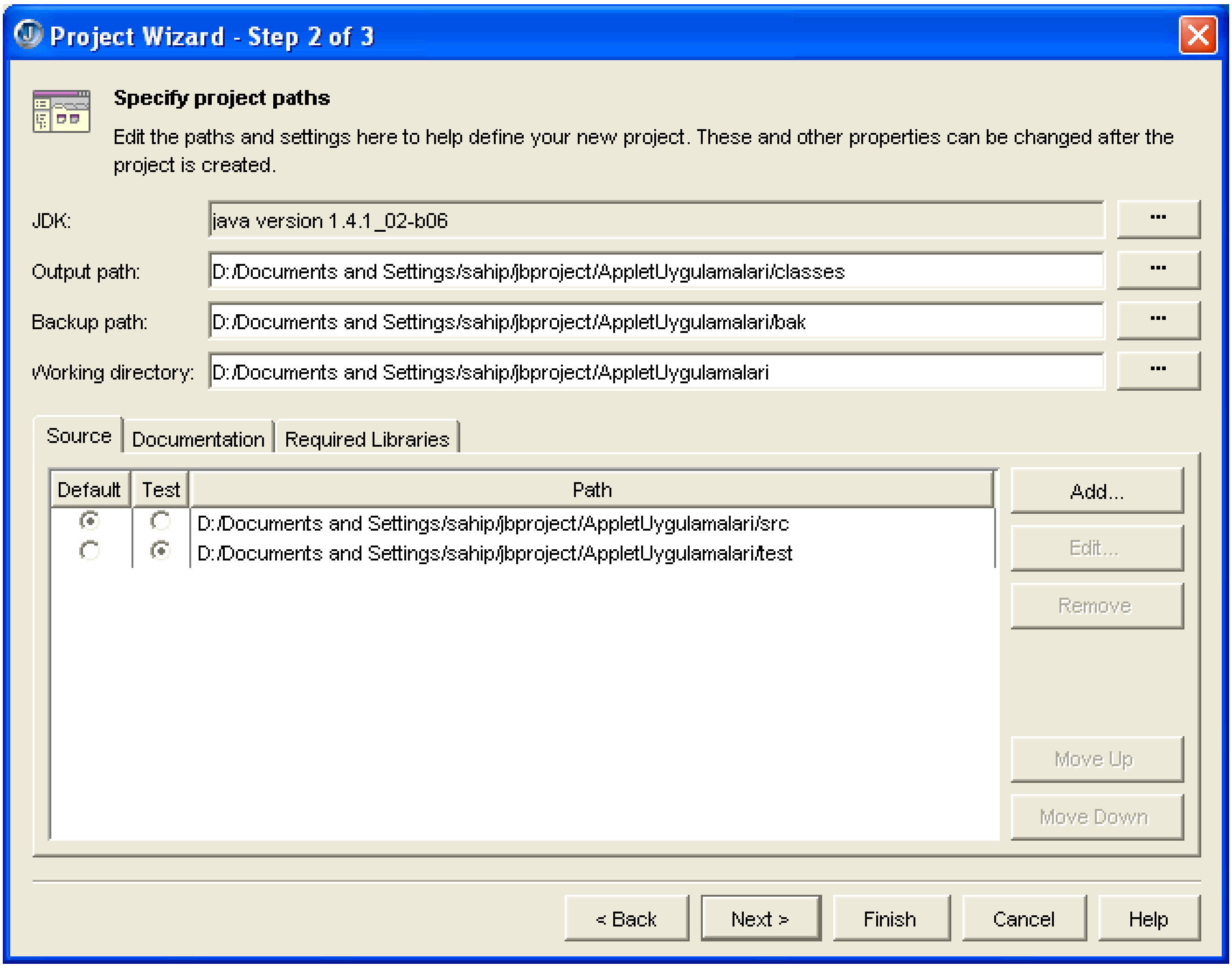
ve bir Applet' i örnek olarak geliştirmem için bulunmaz fırsattı. Bir an bile düşünmeden bu fırsatı değerlendirmem gerektiğine karar verdim.

O gün ilk yaptığım, Help dosyalarından Tutorial' lara bakmak olmuştu. Hedefim, bir Applet' i gönül rahatlığıyla ve kolayca geliştirebilmek ve görsel yazılım geliştirme ortamlarının tüm nimetlerinden faydalanabilmekti. Bu amaçla, bir Applet' in nasıl tasarlandığının ve programlandığının anlatıldığı Tutorial' ı bir solukta okuyuverdim. Ardından hemen uygulamaya geçtim. İlk olarak bana bir proje lazımdı. Vs.Net' teki Solution kavramının karşılığı JBuilder için Project' di. **File-> New Project** sekmesini seçtiğimde, yeni bir proje oluşturmam için gerekli adımları sağlayan bir sihirbaz ile karşılatım.

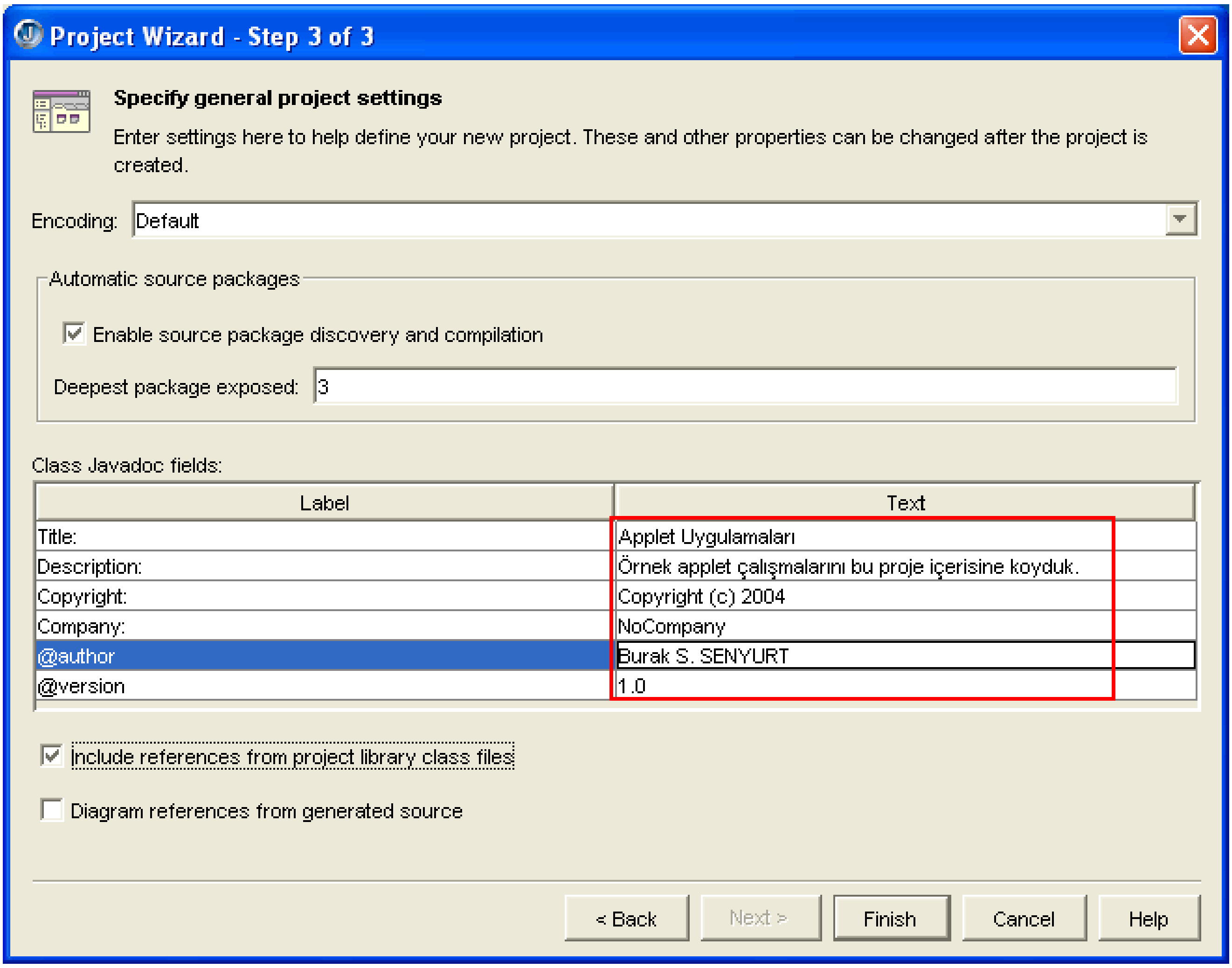


Bu ilk adımda tek yaptığım, Proje için bir isim belirlemek oldu. Hemen ikinci adıma geçtim. İkinci adımda en önemli kısımlardan birisi, JDK isimli yerdi. Burada geliştireceğim uygulamayı Java' nın hangi geliştirme kiti ile yazacağımı belirtebilmekteydim. Output Path ile java dosyalarının derlenmesi sonucu oluşturulacak sınıf dosyalarının hangi yolda

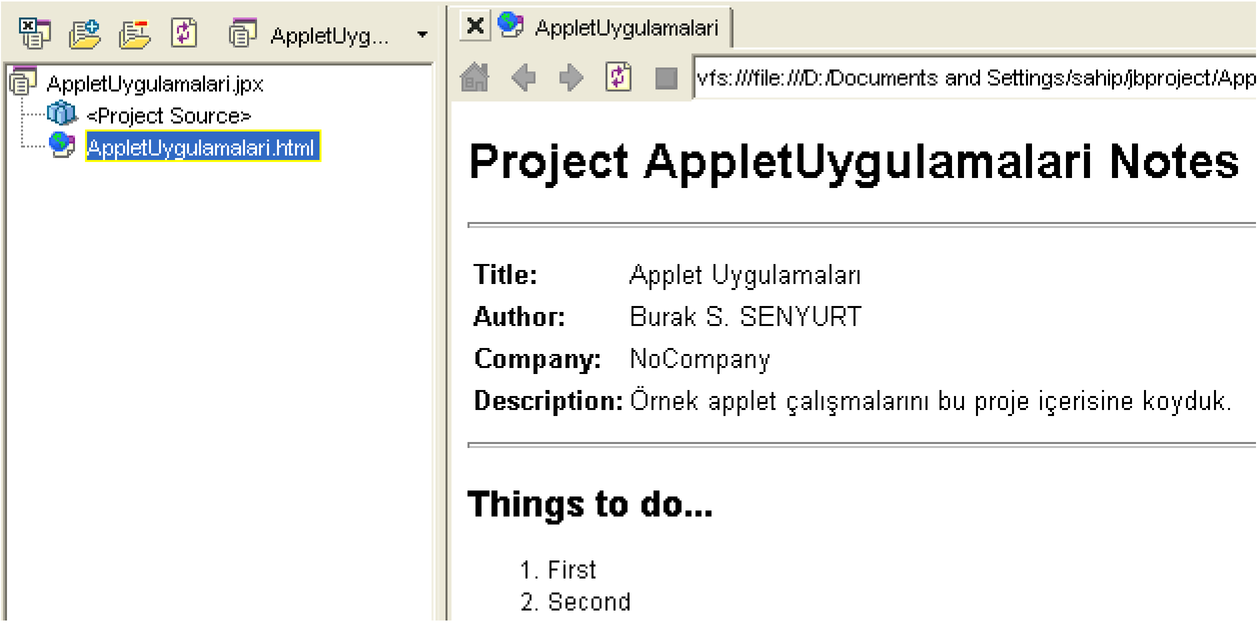
olacağı, Backup Path ile yedeklemelerin nereye yapılacağı belirleniyordu. Working Directory' yi henüz tam olarak kestirememiştim. Bunun yerine projeyi tamamladığımda bu klasörü inceleyip ne olup bittiğine bakmaya karar verdim.



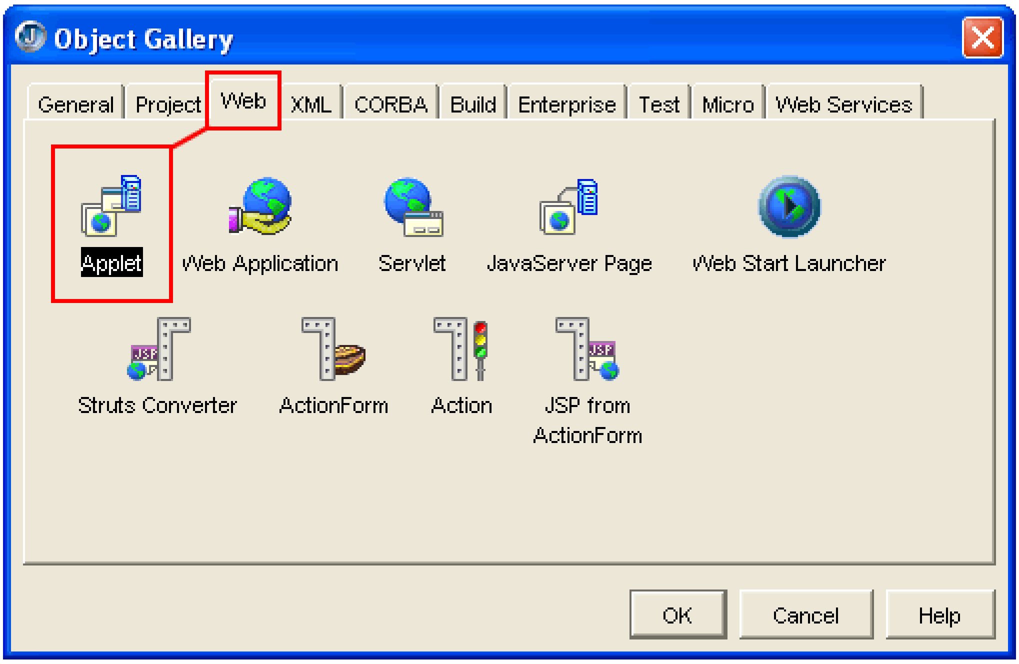
Bu adımıda geçtikten sonra sıra proje oluşturmanın son adımına gelmişti. Bu adımda, proje hakkında bir takım bilgileri girdim. Örneğin projenin adını, kısa bir açıklamasını, versiyon numarasını vs... Tutorial' dan öğrendiğim kadarı ile, bu adımda belirtilenler html bazlı bir dosyada toplanıp, proje hakkında bir takım temel bilgileri sağlamaktaydı.



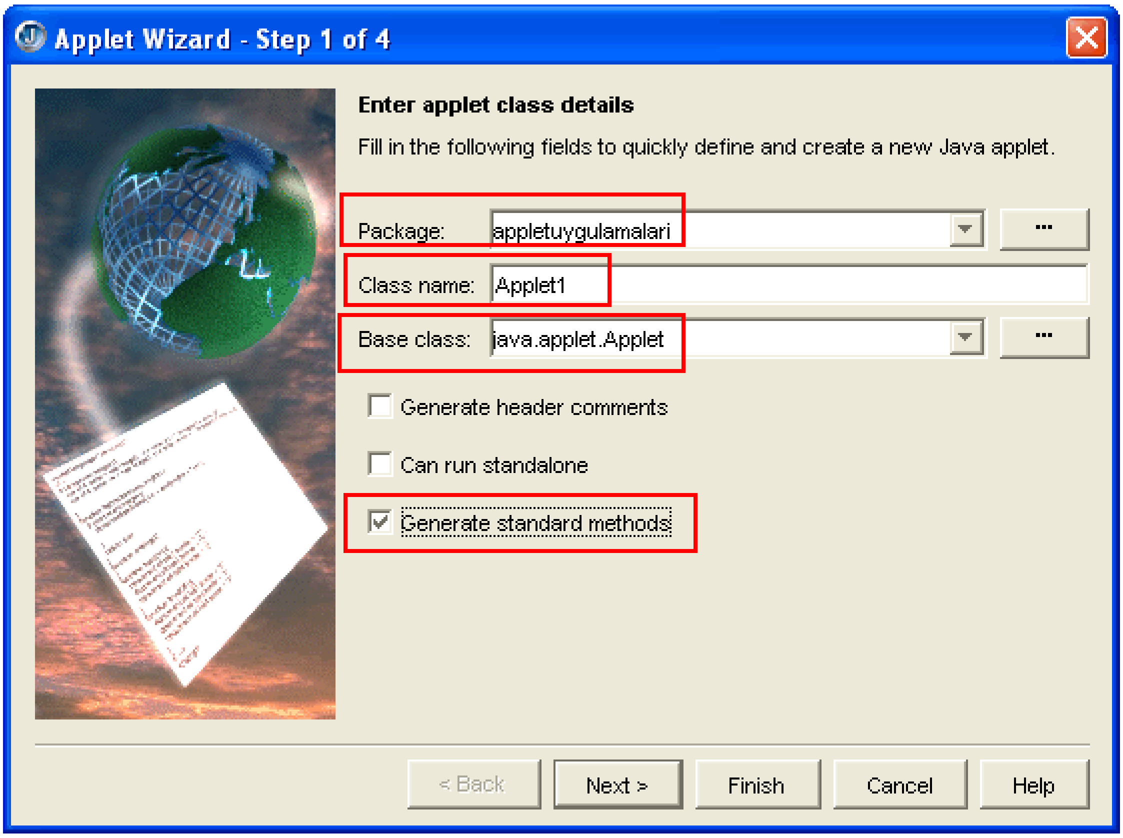
Son olarak Finish diyerek projenin oluşturulmasını sağladım. Proje oluşturulduğunda, son adımda girdiğim verileri içeren bir html dökümanınında otomatik olarak yazıldığını gördüm.



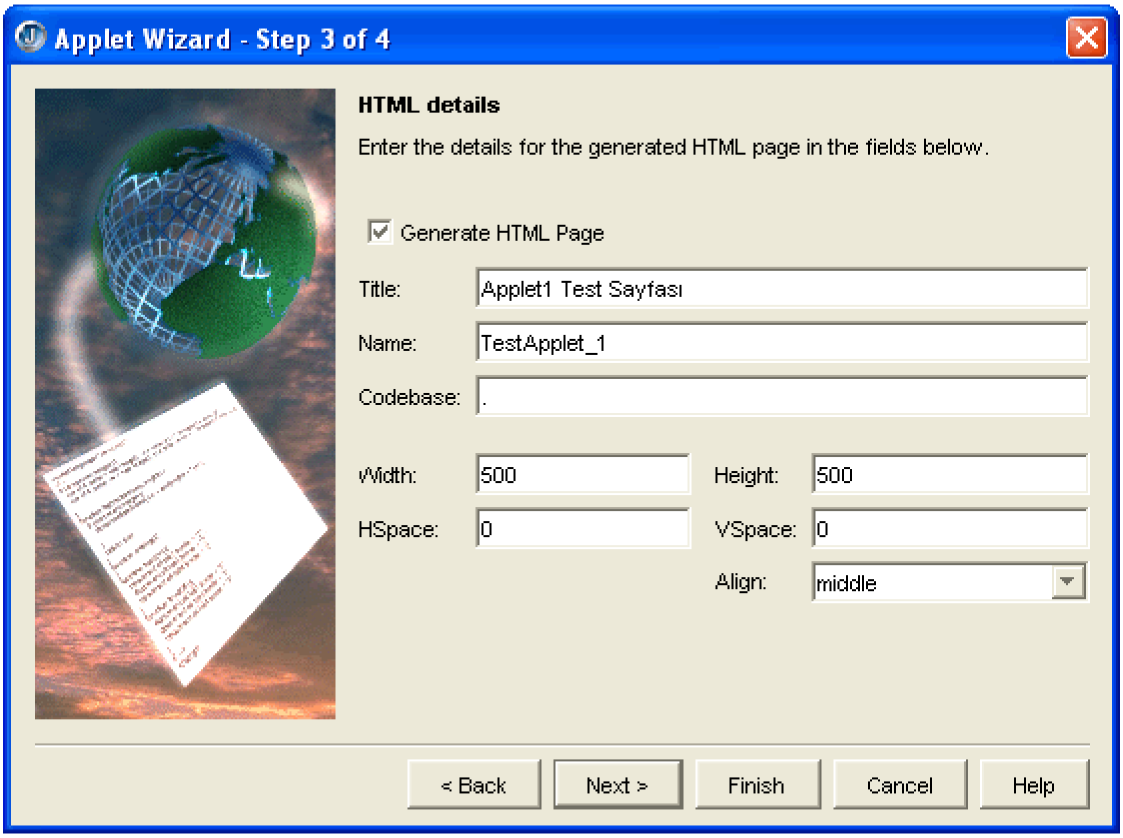
Artık projem hazır olduğunda göre, bu projeye bir Applet ekleyebilirdim. Tek yapmam gereken **File->New** menüsünden açılacak olan **Object Gallery**' de **Web** sekmesine gelmek ve **Applet** şablonunu seçmekti.



Bu işlemin ardından Applet ile ilgili bir takım ayarlamaların yapıldığı başka bir sihirbaz ile karşılaştım.



İlk adımda herşey gayet açık ve netti. Appletin yer alacağı paketin adı standart olarak, proje ismiydi. Applet için bir sınıf adı belirtilmişti. Bu sınıf adını şu an için değiştirmedim. Base class ile, bir appletin türetilmesi gereken Applet sınıfını tanımlamıştım. Özellikle dikkatimi çeken nokta **Generate standart methods** seçeneğidi. Varsayılan olarak işaretli olmayan bu seçeneği işaretledim. Çünkü bu seçenek ile, Applet için gerekli standart bir takım metodların (init gibi) otomatik olarak oluşturulacağını düşünüyordum. Sihirbazda bir sonraki adım Applet' in dışarıdan parametre alıp alamayacağı ile ilgiliydi. Şu an için Applet' in html sayfasından parametre almayacağını düşündüğüm için bu adımı geçerek üçüncü adıma geldim. Üçüncü adımda, Applet'in test edilmesi için oluşturulacak html dökümanına ait bilgiler yer alıyordu.



Burada, HTML dökümanın başlık bilgisini, dosya adını, Applet' in boyutlarını belirledim. Codebase özelliği ile, Applet' in çalışması için gerekli sınıf dosyalarının bulunduğu klasörler belirleniyordu. Sonraki adımıda değiştirmeden geçtiğimde, JBuilder benim için, Applet dosyasını otomatik olarak oluşturmuş ve gerekli kodları yerleştirmişti. Şu haliyle Applet' in kodları aşağıdaki gibiydi.

```
package appletuygulamalari;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Applet1 extends Applet
{
    private boolean isStandalone = false;

    public String getParameter(String key, String def)
    {
        return isStandalone ? System.getProperty(key,
def):(getParameter(key) != null ? getParameter(key) : def);
```

```
}


public Applet1()
{
}

public void init()
{
    try
    {
        jbInit();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception
{
}

public void start()
{
}

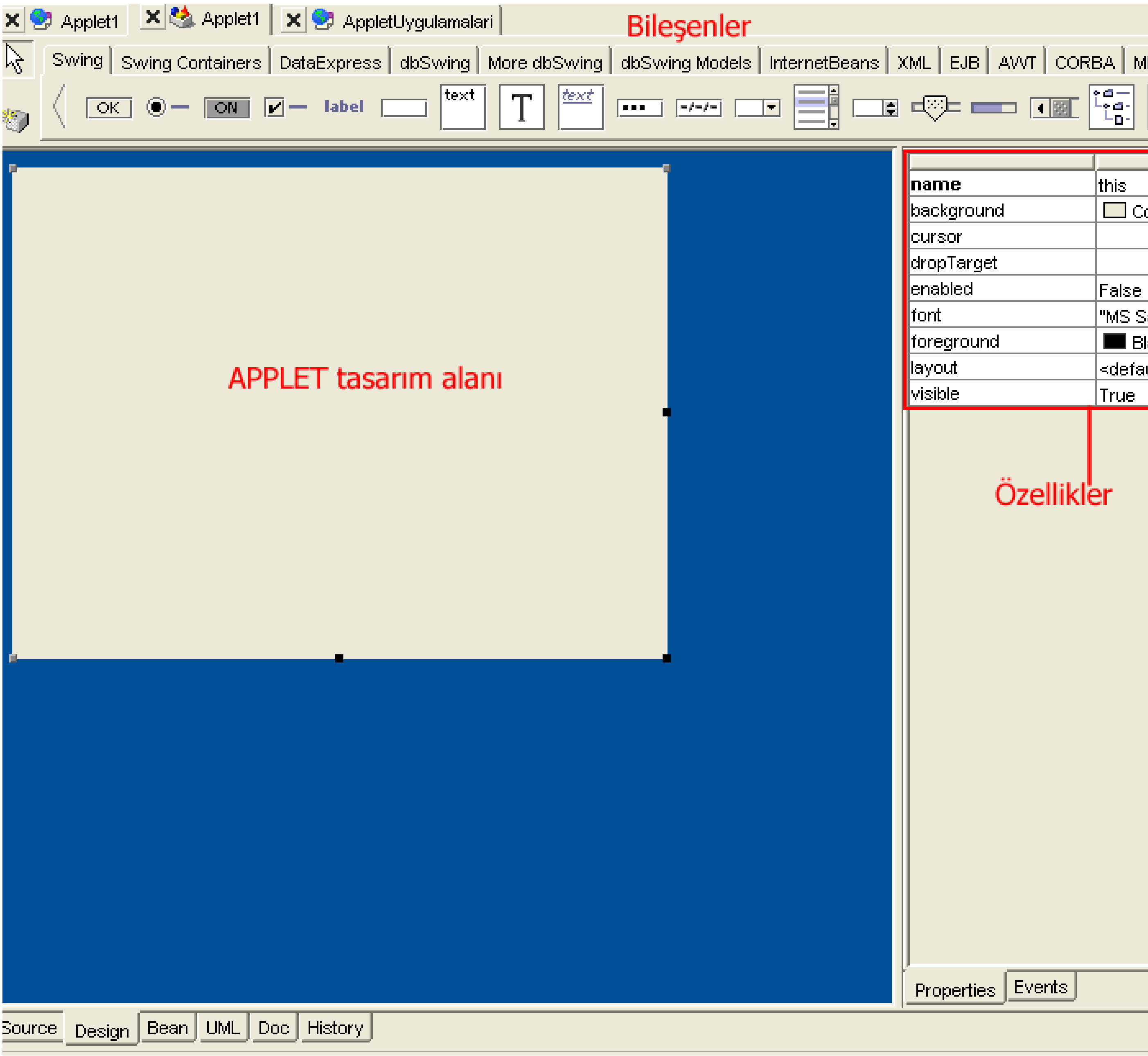
public void stop()
{
}

public void destroy()
{
}

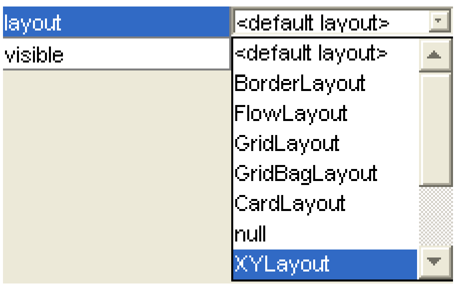
public String getAppletInfo()
{
    return "Applet Information";
}
```

```
    public String[][] getParameterInfo()
    {
        return null;
    }
}
```

Oluşturulan bu Applet dosyası ile, geçen haftalarda oluşturduğum Applet' ler arasında çok fazla fark yoktu. Herşeyden önce, dosyaya baktığımda en azından bir Applet olduğunu, bu nedenle Applet sınıfından türetildiğini biliyordum. Bununla birlikte, olay dinleyicilerinin kullanılabilmesi için ve AWT bileşenlerinin oluşturulabilmesi için gerekli applet, awt ve event paketlerinin referans edildiğinide anlamıştım. Bir Applet' in yaşam süresi içinde devreye giren init, start, stop , destroy gibi metodlarında ne işe yaradığını biliyordum. Ekstradan JBuilder tarafından eklenmiş ilave kodlar vardı. Ancak şu an için bu kodlar pek umurumda değildi. Bir an önce awt bileşenlerini eklemek istiyordum. Bu amaçla hemen design sekmesine geçtim ve işte... Oluşturduğum applet ekranı üst taraftaki koca bir bileşen paleti ile karşımda duruyordu. Sağ tarafta ne olduklarını adım gibi bildiğim özellikler pencereside yüzüme ayrı bir gülümseme katmıştı.



İlk olarak, appletim ile ilgili bir takım özellikleri değiştirdim. Sonuç olarak bu değişikliklerin koda nasıl yansıtılacağını merak ediyordum. Applet' in arka plan rengini değiştirdim. Tam bu sırada dikkatimi layout isimli özellik çekti. Bu özelliğin aldığı bir takım değerler vardı.



Layout' ların, AWT bileşenlerinin Applet üzerine nasıl konumlandırılacağını belirlemekte kullanıldığını biliyordum. Hatta geçen haftaki denemelerimde en çok zorlandığım yerleşim sorununu bir Layout nesnesi yardımıyla çözmüştüm. Yinede istediğim gibi olmamıştı. JBuilder, bana bir sürü layout çeşidi sunmaktaydı. Biraz deneme yanılmadan sonra en uygun olananın XYLayout olduğunu tespit ettim. Çünkü XYLayout sayesinde, AWT bileşenlerini Applet üzerinde serbest olarak istediğim yere konumlandırabilecektim. Yaptığım bu ayarlamalardan sonra kodlarıma baktığımda aşağıdaki değişiklerin olduğunu farkettim.

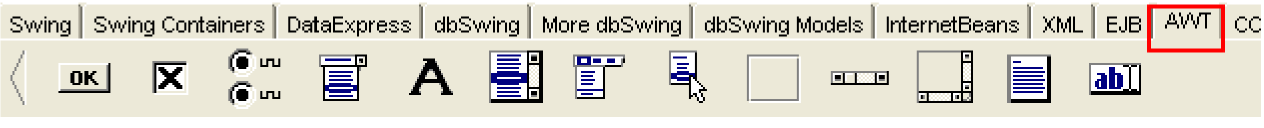
Herşeyden önce,

```
XYLayout xYLayout1 = new XYLayout();
```

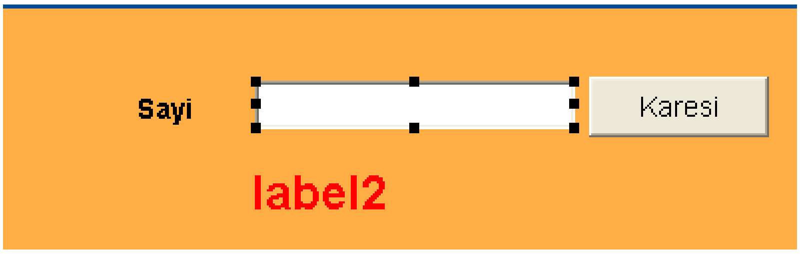
kod satırları ile XYLayout nesnesi oluşturulmuştu. Ancak bu Layout nesnesi, java geliştirme kiti ile gelen sınıflardan değildi. Onun yerine, com.borland.jbcl.layout paketinde yer alan Layout sınıflarından birisiydi. Zaten bu nedenle, bu paket Applet' in başında uygulamaya import edilmişti. Diğer yandan jbInit metodu içinde, bu XYLayout' un Applet' e uygulanması için aşağıdaki kod satırının eklenmiş olduğunu gördüm.

```
this.setLayout(xYLayout1);
```

Artık Applet' im içim seçtiğim Layout desenide ayarlandığına göre gönül rahatlığıyla, AWT bileşenlerini oluşturabilirdim. Tek yapmam gereken bileşen paletinden AWT sekmesine gelmek bir bileşen çizmek ve bu bileşeni mouse ile Applet' in çalışma alanı üzerine çizmekti.



Bbir kaç basit bileşeni Applet üzerine yerleştirdikten ve bunların belli başlı özelliklerini Properties penceresinden değiştirdikten sonra ise bu özelliklerin Applet dosyasına nasıl yansıtıldığını incelemeye başladım.



Applet üzerinde iki Label bileşeni bir TextField bileşeni ve bir de Button bileşeni vardı. Bu bileşenlerin bazı özelliklerini değiştirdiğimde kodun içerisinde JBuilder' ın aşağıdaki eklemeleri yaptığını farkettim. İlk olarak Applet kodlarının başında, her bir AWT bileşeni için birer nesne örneği oluşturulduğunu gördüm.

```
TextField tfSayi = new TextField();
Label label1 = new Label();
Button btnKare = new Button();
Label label2 = new Label();
```

Bu nesnelerin ayarladığım özellikleri ise, jbInit isimli metod içerisine yazılmıştı. Herhangibir bileşenin özelliklerini ayarlamak için Properties penceresini kullanmıştım. Tek satır kod yazmadan, JBuilder, bu işlemlerimi jbInit isimli metodu içerisine otomatik olarak yazıvermişti.

```
private void jbInit() throws Exception
{
   tfSayi.setFont(new java.awt.Font("Dialog", 0, 14));

   this.setBackground(new Color(255, 173, 69));
   this.setLayout(xYLayout1);

   label1.setFont(new java.awt.Font("Dialog", 1, 14));
   label1.setText("Sayi");

   btnKare.setFont(new java.awt.Font("Dialog", 0, 14));
   btnKare.setLabel("Karesi");

   label2.setFont(new java.awt.Font("Dialog", 1, 24));
   label2.setForeground(Color.red);
   label2.setText("label2");
```

```
    this.add(tfSayi, new XYConstraints(128, 36, 160, 24));
    this.add(btnKare, new XYConstraints(295, 34, 90, 30));
    this.add(label1, new XYConstraints(69, 36, 46, 26));
    this.add(label2, new XYConstraints(125, 74, 262, 37));
}
```

TextField bileşeninin Font ayarlamaları setFont metodu ile yapılmıştı. Aynı metod diğer bileşenlerde de kullanılmıştı. Applet' in arka plan rengi setBackground metodu kullanılarak belirlenmekteydi. Bu metoda parametre olarak Color sınıfından bir nesne aktarılmış ve bu nesde R,G,B (Red,Green,Blue) renk tonlarını esas alarak arka planı renklendirmişti. Bileşenlerin başlıklarında yazacak metinler setText veya setLabel metodları ile belirlenmekteydi.

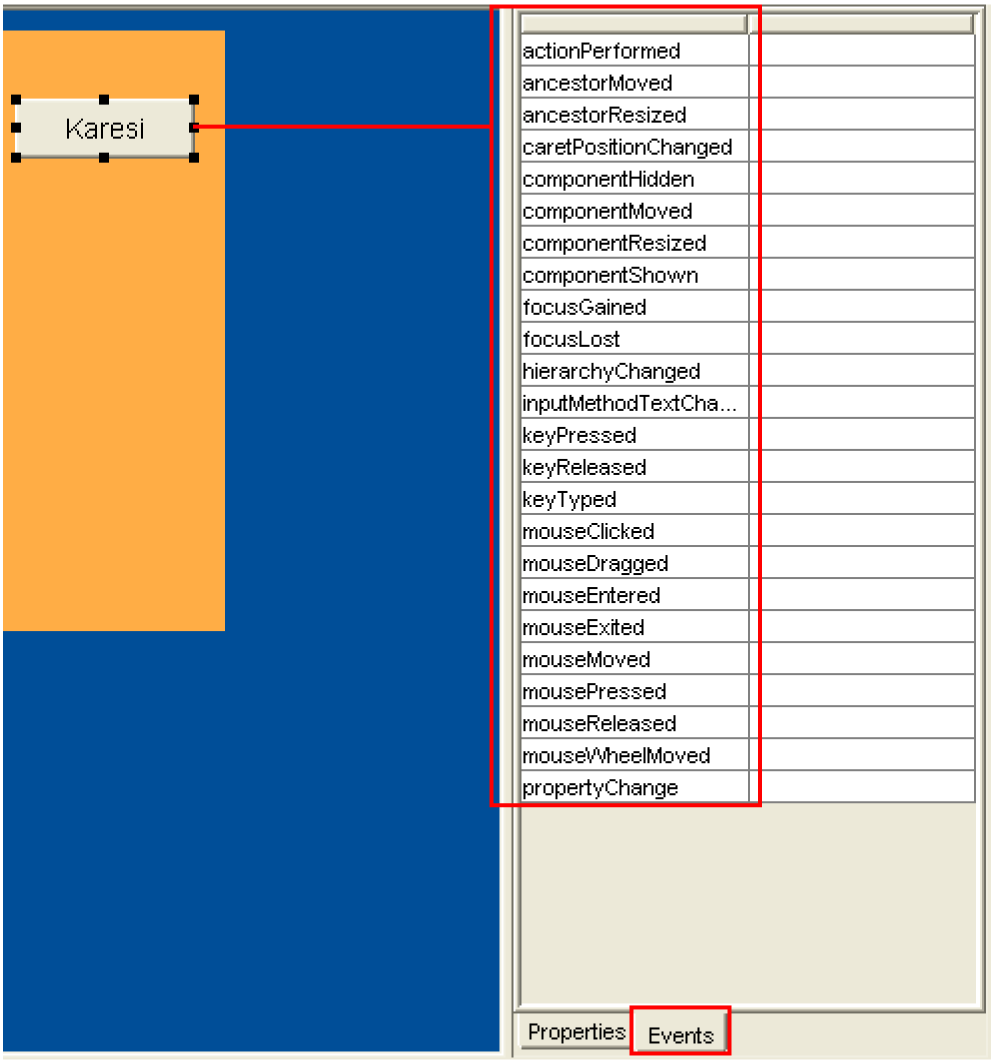
Bu metod içerisinde belkide en önemli kısım, bileşenlerin Applet' e add metodu ile ekleniş şekilleriyle ilgiliydi. Bu metod iki parametre almıştı. İlk parametre ile, hangi bileşenin ekleneceği belirleniyordu. İkinci parametre ise XYConstraints sınıfından bir nesne almaktaydı. Bu nesnenin 4 parametresinden ilk ikisi bileşenin boyutlarını son ikisi ise X ve Y koordinatlarını belirtmekteydi. XYConstraints sınıfıda, XYLayout sınıfı gibi, com.borland.jbcl.layout paketinde yer alan bir sınıftı. Elbette add metodunun bu şeklinin uygulanması, bir XYLayout nesnesinin bu Applet için uygulanmasınıda gerektirmekteydi.

jbInit metodu, bileşenlerin özelliklerinin belirlenmesi ve Applet' e eklenmesinde görev alıyordu. Normalde Notepad ile yazılan bir Applet' te bu kod satırlarını init metodu içerisine doğrudan yerleştirecektim. Nitekim JBuilder bu işi daha güvenli bir hale getirmek ve bileşenlerin Applet' e eklenmesi sırasında oluşabilecek istisnaları kontrol altına alabilmek için init metodunu aşağıdaki gibi düzenlemişti.

```
public void init()
{
    try
    {
```

```
      jbInit();
   }
   catch(Exception e)
   {
      e.printStackTrace();
   }
}
```

Sırada ise Button bileşenine basıldığında gerçekleştirilecek olayların kodlanması vardı. Bu amaçla ilk yaptığım, Button bileşenine çift tıklamak oldu. Olay metodlarının bu şekilde eklendiklerinde, o bileşen için varsayılan olay metodunun oluşturulacağını biliyordum. Olay metodu eklemesi için elbetteki Events sekmesinide kullanabilirdim. Örneğin, Button bileşeni ile kullanabileceğim olay metodları aşağıdaki şekilde görülenlerdi.



Sonuç olarak, JBuilder aşağıdaki gibi bir olay metodu oluşturdu.

```
void btnKare_actionPerformed(ActionEvent e)
{

}
```

Normal şartlarda, bir Button bileşeni için olay metodu oluşturmak isteseydim, aşağıdaki gibi bir metod yazmam gerekicekti. Bu metod içinde, Applet içinde actionPerformed olayına cevap verebilecek bir kaç bileşen olacağını düşünerekten, hangi bileşen için bu olay metodunun çalıştırılacağına ve ne şekilde çalışacağına karar vermek amacıyla, ActionEvent parametresinin getSource metodunnu kullanmam gerekiyordu. Oysaki JBuilder bunu yapmamıştı. Onun yerine sanki sadece bu Button bileşeni için çalışacak bir olay metodu yazmıştı.

```
public void actionPerformed(ActionEvent e)
{
  if(e.getSource()==btnYaz)
  {

  }
}
```

Peki JBuilder bu durumu nasıl gerçekleştirmişti? Kodları incelemeye devam ettim. actionPerformed olay metodu için, ActionListener arayüzünün Applet sınıfına uygulanmış olması gerekiyordu. Oysaki Applet' e ait sınıf tanımlamasında böyle bir arayüz uygulanmamıştı.

```
public class Applet1 extends Applet {
```

Tam bu sırada Applet'e ikinci bir sınıfın daha eklendiğini gördüm.

```
class Applet1_btnKare_actionAdapter implements
java.awt.event.ActionListener
{
   Applet1 adaptee;
```

```
    Applet1_btnKare_actionAdapter(Applet1 adaptee)
    {
        this.adaptee = adaptee;
    }
    public void actionPerformed(ActionEvent e)
    {
        adaptee.btnKare_actionPerformed(e);
    }
}
```

Asıl bu sınıf ActionListener arayüzün uygulamıştı. Sınıfın adı, Applet ve Bileşen adlarından oluşturulmuştu. Sınıf içinde ilk satırda, Applet1 sınıfından (ki bu appletimin adı oluyordu) bir nesne tanımlanmıştı. Sınıfın yapıcı metodunda da bu Applet1 nesnesi kullanılmıştı. Peki ama bunlar ne anlama geliyordu. Sorunun cevabı izleyen actionPerformed metodunda yatmaktaydı. Button bileşenine basıldığında, Applet' in olay dinleyicisi bu yeni sınıf içerisinde yer alan actionPerformed metodunu devreye sokuyordu. Bu metod ise, ActionEvent parametresini, button bileşeni için oluşturulan btnKare_actionPerformed metoduna göndermekteydi.

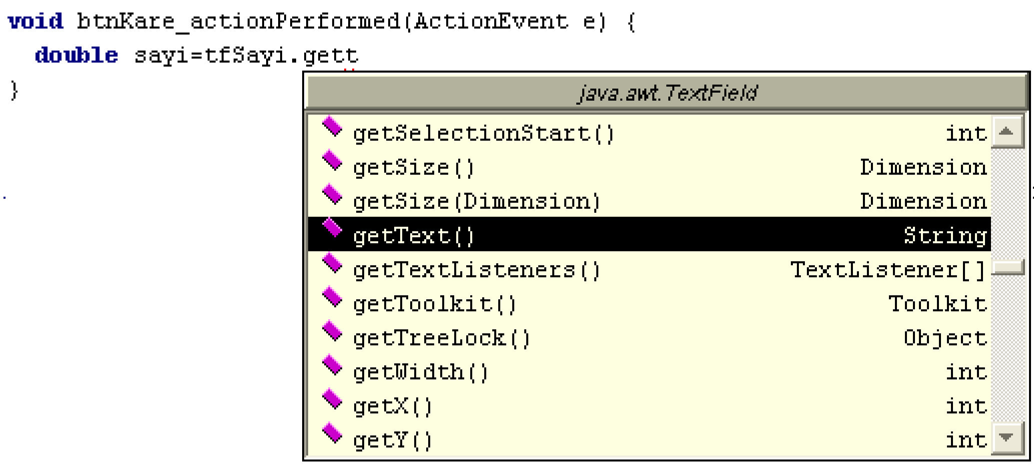
Bu tasarım deseni sayesinde, her bileşen için yazılacak olay metodları ayrı ayrı olacak şekilde oluşturulmaktaydı. Böylece bileşen olaylarının tek bir metod içinden ayrıştırılarak ele alınmasının önüne geçilmiş oluyordu. Ancak yinede eksik bir şeyler vardı. Sonuç olarak bir şekilde, olay dinleyicisinin bir yerlerde bu Button bileşeni için eklenmiş olması gerekirdi. Kodu bir kere daha inceledikten sonra, JbInit metodunda aşağıdaki satırı farkettim.

```
btnKare.addActionListener(new
Applet1_btnKare_actionAdapter(this));
```

Normalde, bu Button bileşeni için olay dinleyicisi eklenirken addActionListener metodu sadece this parametresini alırdı. Burada ise, olay dinleyicisi olarak, Applet1_btnKare_actionAdapter sınıfı türünden bir nesne oluşturulmuş ve bu nesneye this parametresi yani Applet

sınıfının kendisi gönderilmişti. Bir başka deyişle olay dinleme görevini bu yeni sınıf üstlenmekteydi.

Artık Button bileşenine basıldığında neler olacağını kodlayabilirdim. Burada çok daha güzel bir şey farkettim. Borland uzun süredir, geliştirme ortamlarında intellisense özelliğini kullanıyordu. Bu özelliğin JBuilder içindede olması gerçekten çok iyiydi. Nitekim . işaretinden sonra ilgili nesne ile kullanabileceğim metodlar aşağıdaki gibi karşıma geliyordu. Tek gördüğüm eksik, bir metod adı üstüne geldiğimde, Vs.Net' te olduğu gibi metod hakkında kısa bir açıklamanın ekrana gelmiyor olmasıydı.

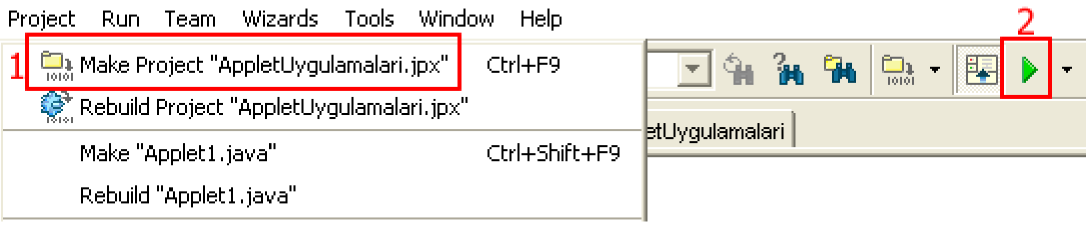


Buna rağmen kodları yazmak kolaydı. Bu mutluluk ve huzur içinde aşağıdaki kodları oluşturdum. Kodlarda gerçekleşen olay son derece basitti. Güvenli bir try-catch bloğu içinde, önce TextField' a girilen değeri getText metodu ile almıştım. Ardından, bu değeri double tipinden bir değişkene aktarmak için Double sınıfının parseDouble metodunu kullandım. Sonra Math sınıfının pow metodu ile bu değerin karesini buldum ve sonucu String sınıfının valueOf metodu ile String türünden bir değişkene aktardım. Son olarak bu değeri label bileşeninde setText metodu ile gösterdim.

```
void btnKare_actionPerformed(ActionEvent e)
{
   try
   {
      String sayi = tfSayi.getText();
      double deger=Double.parseDouble(sayi);
```

```
        deger=java.lang.Math.pow(deger,2);
        String sonuc=String.valueOf(deger);
        label2.setText(sonuc);
    }
    catch(Exception hata)
    {
        label2.setText(hata.toString());
    }
}
```

Tüm bu işlemlerden sonra tek yapmam gereken, projeyi derlemek ve çalıştırmaktı.



Sonuç ise aşağıdaki gibi oldu. Applet başarılı bir şekilde çalıştı. Bu mutluluğu yaşarken güneşin çoktan battığını ve ofisin kapanmak üzere olduğunu farkettim. Masamda 3 kulplu bardak içi boş ve hoş bir kahve kokusu ile duruyordu. Java dili ile birşeyler geliştirmenin bu kadar eğlenceli ve güzel olacağını tahmin etmemiştim. Ancak elbetteki dilin temellerini bilmeden yaptıklarımdanda bir şey anlayamaz ve beceremezdim. Artık rahat ve huzurlu bir şekilde evime dönebilirdim.



Bölüm 17: Layout

Geçtiğimiz hafta hayatımın en mutlu günlerinden birisini, değerli bir arkadaşımının ofisinde, bilgisayarının başında JBuilder kullanarak geçirdim. Bu hafta, o günü mumla aradığımı söylemek isterim. JBuilder ile daha önce hiç uygulama geliştirmemiş olmama rağmen, kolayca adapte olmuştum. Elbetteki benim için en büyük rahatlığı, Applet gibi görsel uygulamaların ekran tasarımlarının son derece kolay bir şekilde yapılabilmesiydi. Bu hafta yine sevimsiz notepad editorüm ile başbaşayım. Aslında amacım arkadaşımın bilgisayarında JBuilder ile başka çalışmalarda yapmaktı. Fakat kendisi tatile çıktığı için banada notepad ile bir kahve molası geçirmek kaldı.

Özellikle notepad editorünü kullanarak applet tasarlamanın en zor yanlarından birisi, applet üzerindeki bileşenlerin yerleşim şekillerinin ayarlanmasının zorluğudur. Bu hafta ne yapıp edip, bu fobiyi yenmeye karar verdim ve java dilinde Layout

kavramını incelemeye başladım. Layout' lar applet üzerine yerleştirilecek bileşenlerin belli bir nizamda olmasını sağlamaktadırlar. Java paketiyle gelen Layout sınıfları 5 adettir.

Java Layouts
GridLayout
BorderLayout
FlowLayout
CardLayout
GridBagLayout

Öncelikle işe en kolay olanından başladım. FlowLayout. Layout sınıflarını anlamanın en iyi yolu elbette onları bir örnek üzerinde uygulamakla mümkün olabilirdi. Bu amaçla çok basit olarak aşağıdaki gibi bir java örneği geliştirdim.

```
import java.awt.*;
import java.applet.Applet;

public class Layouts extends Applet
{
   TextField tf1;
   TextField tf2;
   Button bt1;
   Label lb1;
   Label lb2;

   public void init()
   {
      setLayout(new FlowLayout(FlowLayout.CENTER,15,30));

      lb1=new Label("Username");
      tf1=new TextField(25);
      lb2=new Label("Password");
      tf2=new TextField(25);
      bt1=new Button(" OK ");
```

```
        add(lb1);
        add(tf1);
        add(lb2);
        add(tf2);
        add(bt1);
    }
}
```

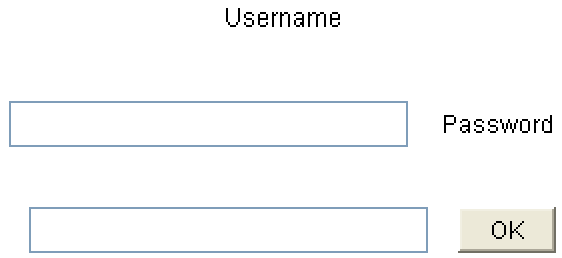
Burada tek yaptığım, applet üzerine yerleşecek bileşenlerin FlowLayout 'a göre düzenlenmesiydi. Applet' in FlowLayout' u uygulayacağını belirtmek için,

```
setLayout(new FlowLayout(FlowLayout.CENTER,15,30));
```

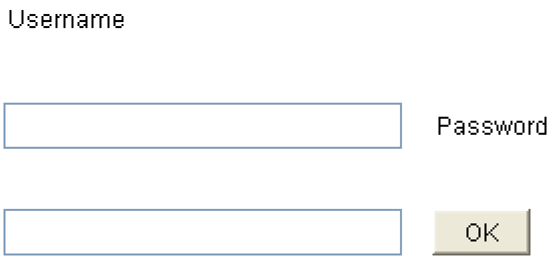
kod satırını kullandım. Burada, bileşenlerin Applet üzerinde bulundukları satırda ortalanarak yerleştirilecekleri ve yatay olarak aralarında 15 piksel, dikey olarak ise 30 piksel boşuk olacağı belirtiliyor. Peki ama bu yerleşim nasıl oluyor? Nitekim kaynaklarda, FlowLayout' un bileşenleri sola ve aşağı doğru hizaladığı söyleniyordu. Denemekten başka çarem olmadığını düşünerek hemen işe koyuldum ve basit bir html sayfasında applet tagımı aşağıdaki gibi ekledim.

```
<APPLET CODE="Layouts.class" width="300" height="500">
</APPLET>
```

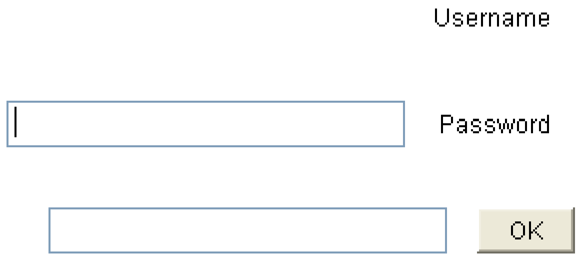
Şimdi bu html sayfasını çalıştırdığımda aşağıdaki gibi bir görüntü elde ettim.



Doğruyu söylemek gerekirse kel alaka bir tasarım olmuştu. FlowLayout için diğer iki durum, yerleşim şeklinin sola dayalı olmasını belirten FlowLayout.LEFT ve sağa dayalı olmasını belirten FlowLayout.RIGHT seçenekleriydi. Önce LEFT durumunu inceledim ve aşağıdaki ekran görüntüsünü elde ettim.



Sanki durumu anlamaya başlamış gibiydim. Bu kez RIGHT durumunu denedim ve aşağıdaki sonucu elde

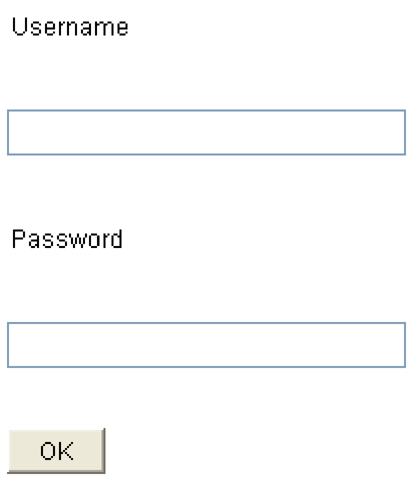


ettim.

Görünen o ki, düzgün bir tasarım yapmak istiyorsam bunu öncelikle kafamda yapmalı ve Applet' in boyutlarını çok tutarlı belirtmeliydim. Ancak bu şekilde düzgün bir tasarım elde edebilirdim. Şu an için deneme yanılma yöntemini kullanmaktan başka bir şey aklıma gelmiyor açıkçası. Örneğin, en uzun Label olan Username ile TextField' ların boyu düşünüldüğünde Applet tagını aşağıdaki gibi düzenlemek daha mantıklı geliyordu.

```
<APPLET CODE="Layouts.class" width="250" height="500">
</APPLET>
```

Java kodunda da, FlowLayout dizilimini FlowLayout.LEFT olarak belirlediğimde, daha düzenli bir ekran görüntüsü elde ettim. Gerçi bu varsayımsal yaklaşım ile kullanılan teknik pek hoşuma gitmemişti ama en azından buz dağının üst kısmını biraz olsun yontmayı başarabilmiştim.



Bu örnekten sonra, arkadaşımı bir kat daha özledim desem yalan olmaz. Heleki JBuilder uygulamasını. Oradaki tasarım rahatlığı gerçektende muhteşemdi. Bu sırada aklıma başka bir şey geldi. Acaba, bir Layout düzeneğini, bir Panel bileşenine uygulayabilir miydim? Eğer böyle bir şey söz konusu olursa, görsel tasarımı biraz daha kolaylaştırabilirdim. Bu amaçla aşağıdaki gibi bir örnek geliştirdim.

```
import java.awt.*;
import java.applet.Applet;

public class Layouts extends Applet
{
   TextField tf1;
   TextField tf2;
   Button bt1;
   Label lb1;
   Label lb2;
   Panel p1;

   public void init()
   {
      p1=new Panel();
      p1.setBackground(Color.yellow);
      p1.setLayout(new FlowLayout(FlowLayout.LEFT));

      lb1=new Label("Username");
      tf1=new TextField(25);
      lb2=new Label("Password");
```

```
        tf2=new TextField(25);
        bt1=new Button(" OK ");

        p1.add(lb1);
        p1.add(tf1);
        p1.add(lb2);
        p1.add(tf2);
        p1.add(bt1);

        add(p1);
    }
}
```

İlk olarak, bir Panel bileşeni oluşturdum ve bu bileşen üzerine yerleştireceğim diğer bileşenlerin FlowLayout düzeneğine göre konumlandırılmalarını sağlamak için Panel bileşenine,

```
p1.setLayout(new FlowLayout(FlowLayout.LEFT));
```

satırındaki setLayout metodunu uyguladım. Böylece, Panel bileşeni üzerine yerleşecek bileşenler, FlowLayot düzeneğine göre, Layout' un solundan hizalanacak şekilde konumlanacaklardı. Bileşenleri Panel' e eklemek için, Panel sınıfına ait add metodunu kullandım. Tabi bu işlemlerden sonra Panel bileşeninide, Applet' e add metodu ile eklemeyi unutmadım. Bu adımlardan sonra, Java dosyasını derleyip applet' i içeren html sayfasını ilk çalıştırdığımda aşağıdaki sonucu elde ettim.



Böyle olacağı belliydi zaten. Applet tagında width özelliğini arttırmam gerekiyordu. Bu değeri 600 olarak belirledim. Şimdi elde ettiğim sonuç çok daha iyiydi.



Layout sınıfları bitmek bilmiyordu. Sırada BorderLayout sınıfı vardı. Bu sınıfın en ilginç yanı, Applet ekranının, NBA basketbol takımlarının liglerinde gruplanışlarına benzer bir yapıda ayrıştırılıyor olmasıydı. Doğu Grubu, Merkez Grubu, Batı Grubu, Kuzey Grubu ve Güney Grubu. Yani, applet üzerine ekleyeceğim bileşenleri, bu gruplara yerleştirmem gerekiyordu. Bunu görsel olarak anlayabilmek için, kaynaklarımı araştırdım ve yukarıdaki örneğe bu kez BorderLayout düzeneğini uyguladım.

```
import java.awt.*;
import java.applet.Applet;

public class Layouts extends Applet
{
   TextField tf1;
   TextField tf2;
   Button bt1;
   Label lb1;
   Label lb2;

   public void init()
   {
      setLayout(new BorderLayout());

      lb1=new Label("Username");
      tf1=new TextField(25);
      lb2=new Label("Password");
      tf2=new TextField(25);
      bt1=new Button(" OK ");

      add("North",lb1);
      add("Center",tf1);
      add("West",lb2);
      add("South",tf2);
      add("East",bt1);
   }
}
```

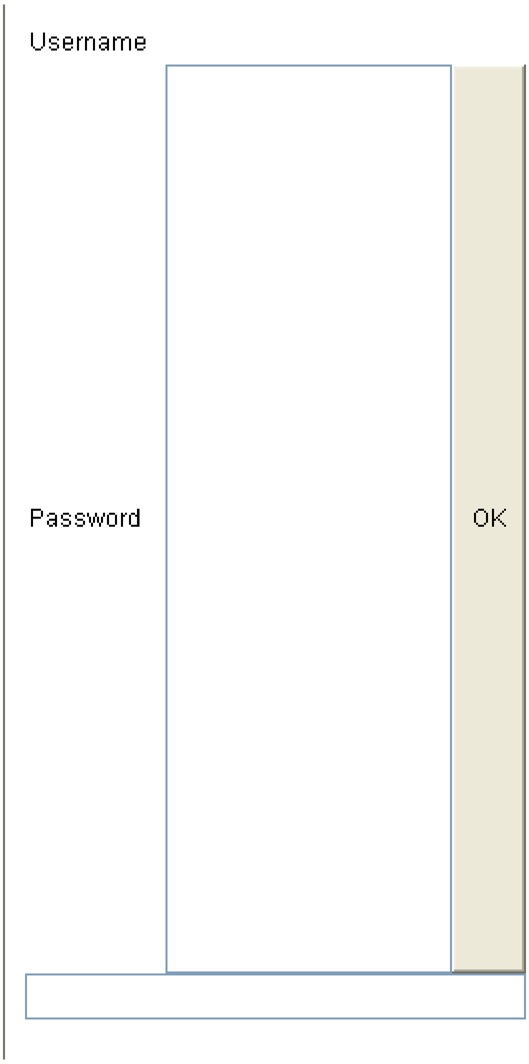
Applet üzerindeki bileşenlerin BorderLayout düzeneğine göre yerleştirileceğini,

```
setLayout(new BorderLayout());
```

satırı belirtiyordu. BorderLayout, Applet' i 5 yön bölgesine böldüğü için, her bileşeni Applet' e eklerken add metoduna ilk parametre olarak, bileşenin hangi bölgeye yerleştirileceğinin belirtilmesi gerekiyordu. Bu işlemde aşağıdaki satırla gerçekleştirmekteydi.

```
add("North",lb1);
add("Center",tf1);
add("West",lb2);
add("South",tf2);
add("East",bt1);
```

Örneğin, tf2 isimli TextField bileşeni Applet'in South (Güney) bölgesine yerleşecekti. Artık Applet' i kullandığım html sayfasını çalıştırıp sonuçları görmek istiyordum. Ama sonuçları göstermeye çekiniyorum açıkçası. BorderLayout' un Applet ekranına uygulanması sonucu aşağıdaki gibi muhteşem ötesi bir tasarım elde ettim. Tarihi bir başarı olduğunu belirtmek isterim.



İşin kötü yanı, belli bir bölgeye sadece tek bir bileşen yerleştirebiliyor olmasıydı. BorderLayout düzeneğinin nerede kullanılacağını düşünürken, diğer Layout' ları incelemenin daha uygun olacağını farkettim. Nitekim, BorderLayout her ne kadar yukarıdaki ekran tasarımı için uygun olmasada elbetteki kullanıldığı bir takım yerler olabilirdi. Layout' lar ile uğraşmak gerçekten insana sıkıcı geliyor. Artık JBuilder gibi görsel tasarım araçlarının o kadar paraya gerçekten deydiğini söyleyebilirim. Hemde gönül rahatlığıyla. Bu düşünceler eşliğinde kendime yeni bir fincan kahve aldıktan sonra bir diğer Layout sınıfı olan GridLayout' u incelemeye başladım. GridLayout ile nispeten biraz daha güzel ve kolay form tasarımları oluşturabileceğimi düşünüyordum. Yapmam gereken son derece basitti. Aynı örneği GridLayout sınıfı ile geliştirmek.

```
import java.awt.*;
```

```
import java.applet.Applet;

public class Layouts extends Applet
{
   TextField tf1;
   TextField tf2;
   Button bt1;
   Label lb1;
   Label lb2;
   Panel p1;
   Panel p2;

   public void init()
   {
      setLayout(new GridLayout(2,1));

      p1=new Panel();
      p1.setBackground(Color.orange);
      p1.setLayout(new FlowLayout(FlowLayout.LEFT));

      p2=new Panel();
      p2.setBackground(Color.blue);

      lb1=new Label("Username");
      tf1=new TextField(25);
      lb2=new Label("Password");
      tf2=new TextField(25);
      bt1=new Button(" OK ");

      p1.add(lb1);
      p1.add(tf1);
      p1.add(lb2);
      p1.add(tf2);
      p1.add(bt1);

      add(p1);
      add(p2);
   }
}
```

Bu kez, Applet'e iki Panel bileşeni ekledim. Bu iki Panel bileşeninde 2 satır ve 1 sütunluk bir GridLayout düzeneği içinde Applet' e ekledim. GridLayout nesneleri, temel olarak iki parametre almakta. İlk parametre, satır sayısını belirtirken, ikinci parametrede doğal olarak sütun sayısını belirtiyor. Bununla birlikte, GridLayout sınıfının 4 parametre alan bir diğer yapıcısıda mevcut. Bu yapıcının aldığı son iki parametre ise, Grid hücrelerine yerleştirilecek bileşenlerin arasındaki yatay ve dikey uzaklıkları piksel olarak belirtmekte. Bu örneği geliştirdikten sonra Applet tagını aşağıdaki gibi düzenledim.

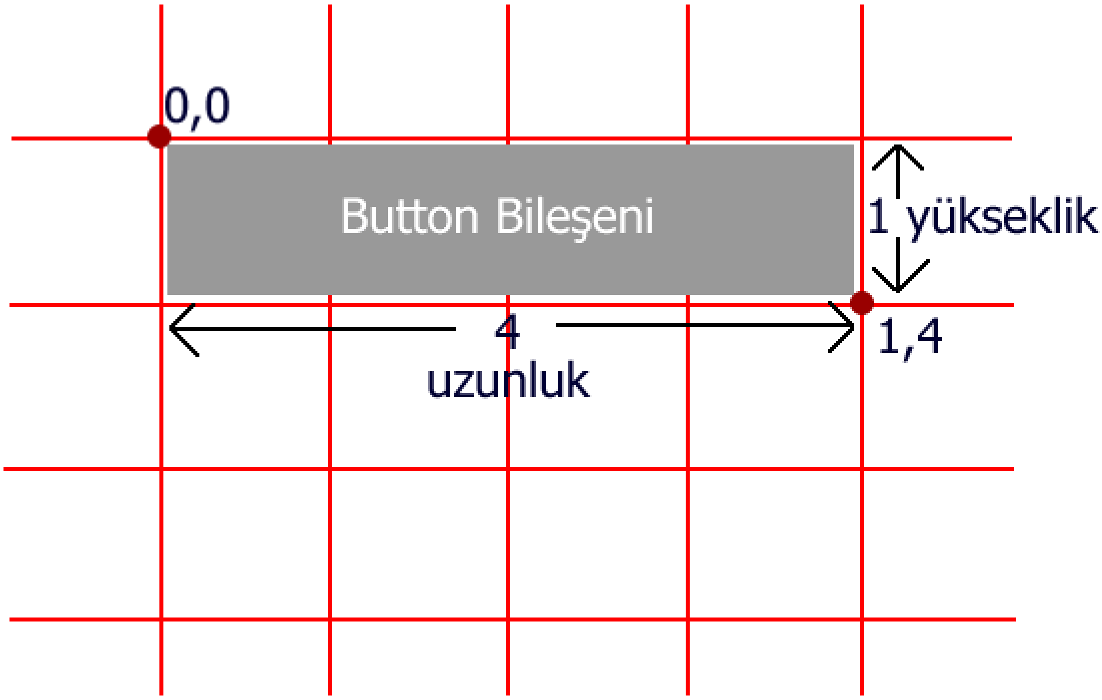
```
<APPLET CODE="Layouts.class" width="610" height="100">
</APPLET>
```

Sonuç ise, FB (FB lilerin şampiyonluklarınıda tebrik ederim bu arada) renkleriyle aşağıdaki gibi oluştu. BJK taraftarı olmama rağmen, tasarımı böyle renklendirebildiğim için çok mutlu oldum. Demek ki, Layout' ları bir arada kullanarak ve işin içine Panel bileşenlerini katarak daha sağlam tasarımlar oluşturulabilirdi. Notepad ile dahi olsa. Ama bununla kim uğraşırdıki güzelim JBuilder varken. (Elbette ben uğraşırdım sanırım.)



GridLayout sınıfınıda geçtim derken karşıma daha karmaşık bir Layout sınıfı çıkıverdi. GridBagLayout. Bunu anlayabilmek için, Bilim Teknik dergisinde her ay çıkan soruları çözer gibi kağıt kaleme sarılıp çalışmam gerektiğini itiraf etmeliyim. GridBagLayout sınıfı ile, Applet üzerinde çok karmaşık ekran tasarımlarını yapabilmek mümkün. Ancak bu işi başarabilmek için bir o kadarda karışık bir teknik kullanmak gerekiyor. Burada önemli olan nokta, Applet üzerine yerleşecek bileşenlerin X,Y koordinatları ile uzunluk ve yükseklik bilgileri. Nitekim, GridBagLayout tekniğinde, Applet karelere bölünüyor ve bileşenler bu kareler üzerine yerleştiriliyor. Şimdi burada hakketten ele kağıt kalem almak ve çizmek lazım. Ancak dijital

bir ortamda olduğumuzu düşünürsek bu iş için, gelişmiş grafik programlarınıda kullanabilirim. İşte bu amaçla bu Layout sınıfını kullanarak, Applet üzerinde konumlandırılacak bir Button nesnesini göz önüne alarak basit bir çizim oluşturdum.



Temel olarak yapacağım işlem buydu. Applet ekranını karelere bölmek. Örneğin bir Button yerleştirmek istiyorum. Önceden bu Button bileşenin X, Y koordinatlarını, uzunluk ve yükseklik bilgilerini, yukarıdaki gibi bir şekli baz alarak bilmem gerekiyor. Olayın esprisi buydu işte. Bu nedenle de GridBagLayout diğer Layout sınıflarına göre daha karmaşık Applet tasarımları oluşturmamıza izin veriyordu. Çünkü bir ekran pozisyonlaması için gereki 4 önemli unsuru baz alıyordu. X,Y Koordinatları, uzunluk ve yükseklik. Peki bu zahmetli tasarımı kodlamada nasıl gerçekleştirebilirdim. Bu amaçla epeyce bir saatimi, sırf GridBagLayout' un nasıl kullanıldığını anlamaya harcadım. Sonuç olarak aşağıdaki gibi bir örnek geliştirdim.

```
import java.awt.*;
import java.applet.Applet;

public class Layouts extends Applet
{
   TextField tf1;
   TextField tf2;
```

```
Button bt1;
Label lb1;
Label lb2;
GridBagLayout gbl;
GridBagConstraints gbc;

public void init()
{

    gbl=new GridBagLayout();
    gbc=new GridBagConstraints();

    setLayout(gbl);
    gbc.insets=new Insets(2,2,2,2);
    gbc.fill=GridBagConstraints.BOTH;

    gbc.gridx=0;
    gbc.gridy=0;
    gbc.gridwidth=5;
    gbc.gridheight=1;
    lb1=new Label("Username");
    gbl.setConstraints(lb1,gbc);
    add(lb1);

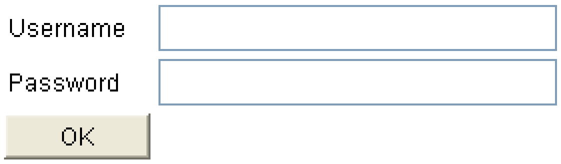
    gbc.gridx=6;
    gbc.gridy=0;
    gbc.gridwidth=5;
    gbc.gridheight=1;
    tf1=new TextField(25);
    gbl.setConstraints(tf1,gbc);
    add(tf1);

    gbc.gridx=0;
    gbc.gridy=1;
    gbc.gridwidth=5;
    gbc.gridheight=1;
    lb2=new Label("Password");
    gbl.setConstraints(lb2,gbc);
    add(lb2);
```

```
        gbc.gridx=6;
        gbc.gridy=1;
        gbc.gridwidth=5;
        gbc.gridheight=1;
        tf2=new TextField(25);
        gbl.setConstraints(tf2,gbc);
        add(tf2);

        gbc.gridx=0;
        gbc.gridy=3;
        gbc.gridwidth=5;
        gbc.gridheight=1;
        bt1=new Button(" OK ");
        gbl.setConstraints(bt1,gbc);
        add(bt1);
    }
}
```

Kodları yazdıktan sonra önce Applet' in nasıl çalıştığına baktım. Sonuç hiçte fena değildi.



Nasıl olmuştuda böylesine güzel bir sonuç elde edebilmiştim. Herşeyden önce, GridBagLayout sınıfını oluşturdum. Ancak burada GridBagConstraints isimli başka bir sınıf daha vardı. Bu sınıf, her bir bileşen için gerekli X, Y koordinatları ile, uzunluk ve yükseklik ayarlamalarını taşıyacak nesneler örneklendirmek amacıyla kullanılmaktaydı. Dolayısıyla bir bileşeni, GridBagLayout sınıfı nesnesine eklerken setConstraints metodu ikinci parametre olarak, GridBagConstraints sınıfına ait nesne örneğini almaktaydı. Böylece, setConstraints metodunun ilk parametresinde belirtilen bileşen, Applet üzerinde belirtilen X, Y koordinatlarına, belirtilen yükseklik ve uzunlukta çiziliyordu. Burada uzunluk ve yükseklik piksel bazında değil hüce bazındadır. Örneğin aşağıdaki satırları göz önüne alalım.

```
gbc.gridx=6;
gbc.gridy=1;
gbc.gridwidth=5;
gbc.gridheight=1;
tf2=new TextField(25);
gbl.setConstraints(tf2,gbc);
add(tf2);
```

Burada TextField bileşeni, X=6, Y=1 koordinatlarına yerleştirilmiştir. Uzunluğu 5 hücre kadar, yüksekliği ise 1 hücre kadardır. Daha sonra, bu bileşen GridBagLayout nesnesine, setConstraints metodu ile eklenmiştir. Artık, bu bileşeni Applet' e add metodu ile eklediğimizde, GridBagLayout konumları esas alınacaktır.

```
gbc.insets=new Insets(2,2,2,2);
gbc.fill=GridBagConstraints.BOTH;
```

Satırlarına gelince. Bu satırlarda, GridBagConstraint nesnesinin bir takım özellikleri belirlenmektedir. Bunlardan birisi insets özelliğidir. Insets ile, hücreler içindeki bileşenlerin hücrenin kenarlarına olan uzaklığı belirtilmektedir. Bir başka deyişle boşluk miktarı. Fill özelliği ileyse, GridBagLayout' un genişlemesi durumunda, hücre içi bileşenlerin her yönde eşit şekilde genişlemesi belirtilmiştir.

Bu zor ama güçlü Layout' tan sonra, incelemeyi unuttuğum bir Layout daha olduğunu farkettim. CardLayout. İlk başta iskambil desteleri ile bir alakası olabilir mi diye düşündüm. Gerçektende öyleymiş. CardLayout' ta, birbirlerinin üstüne binen katmanlar söz konusu. İşin güzel yanı ise, CardLayout' lar ile, birbirinden farklı katmanların tasarlanabilmesi ve çalışma zamanında bu katmanlardan sadece birisinin görünür olması. Dolayısıyla bu katmanlar arasında gezinmek için, olay güdümlü programlama tekniklerini kullanmak gerekiyor. Nasıl mı? İşte örnek.

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
```

```
public class Layouts extends Applet implements ActionListener
{
   TextField tf1;
   TextField tf2;
   Button bt1;
   Button bt2;
   Label lb1;
   Label lb2;
   Label lb3;
   Panel p1;
   Panel p2;
   Panel p3;
   Panel p4;
   CardLayout iskambil;

   public void init()
   {

      iskambil=new CardLayout();

      p1=new Panel();
      p2=new Panel();
      p2.setBackground(Color.blue);
      p3=new Panel();
      p3.setBackground(Color.green);
      p4=new Panel();
      iskambil=new CardLayout();
      p4.setLayout(iskambil);

      lb1=new Label("Username");
      p1.add(lb1);

      tf1=new TextField(25);
      p1.add(tf1);

      lb2=new Label("Password");
      p1.add(lb2);

      tf2=new TextField(25);
      p1.add(tf2);
```

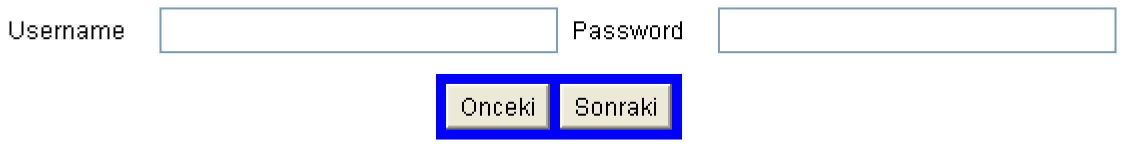
```
        bt1=new Button("Onceki");
        bt1.addActionListener(this);
        bt2=new Button("Sonraki");
        bt2.addActionListener(this);

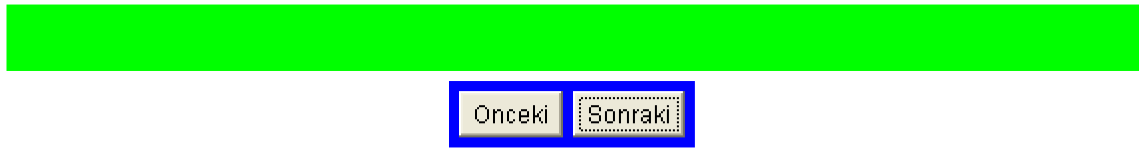
        p2.add(bt1);
        p2.add(bt2);

        p4.add("Giris",p1);
        p4.add("Diger",p3);

        add(p4);
        add(p2);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(e.getActionCommand()=="Onceki")
        {
             iskambil.previous(p4);
        }
        else if(e.getActionCommand()=="Sonraki")
        {
            iskambil.next(p4);
        }
    }
}
```

Ve sonuç. CardLayout daha güzel bir tasarım sundu. Tek bir applet sayfasında üst üste duran paneller arasında gezebilme imkanı.





Artık Layout' lara veda etme vakti geldi. Gelecek kahve molalarında umarım JBuilder ile daha fazla ilgilenme fırsatı bulabilirim. Katetmem gereken daha çok kilometre taşı var. Network programlama, veritabanları, web servisleri, swing bileşenleri, windows uygulamaları vs...

Bölüm 18: Pencereler

Bir kaç haftadır Java dilini popüler yapan Applet� ler ile uğraşıyorum. Buna karşın günümüz dünyasının bir programlama dilinden bekledikleri arasında mutlaka windows uygulamalarının var olması gerektiğini düşünüyorum. Sonuç olarak Applet� ler her ne kadar çok başarılı olsalarda, zaman zaman windows uygulamaları geliştirmemizde gerekiyor. Bir windows uygulamasının belkide en temel özelliği mutlaka bir Form (Nam-ı diğer pencere diyebiliriz) ekranına sahip olması. Peki java dilinde, windows uygulamları oluşturmak için nasıl bir yol izlemem gerekir. İşte bu hafta boyunca, java dili ile bağımsız olarak çalışabilen pencereleri incelemeye çalıştım.

Sun� ın Java paltformu, Microsoft� un ciddi rakiplerinden birisi. Belkide tek ciddi rakibi. Ancak bu rekabet zaman zaman biraz komik olaylarada neden olmuyor değil. Örneğin, yaptığım araştırmalarda gördüm ki, Windows uygulamarında Form kavramı, java dilinde Frame olarak adlandırılıyor. Bu kısa politik düşüncelerden sonra, artık ilk form ekranımı, pardon düzeltiyorum; ilk frame ekranımı tasarlamam gerektiğine karar verdim. Bu amacımı gerçekleştirebilmek amacıyla aşağıdaki çok kısa uygulamayı yazdım.

```
import java.awt.*;

public class IlkPencere
{
   public static void main(String args[])
   {
      Frame pencere=new Frame("ILK PENCEREM");
      pencere.setLocation(0,0);
```

```
        pencere.setBackground(Color.red);
        pencere.setVisible(true);
    }
}
```

Yazdığım bu java dosyasının derledikten sonra çalıştırdım. Karşımda beni bekleyen güzel bir pencere olacağı düşüncesindeydim. Gerçektende muazzam bir pencere oluşturmayı başarmıştım :)



Doğruyu söylemek gerekirse daha büyük bir frame olacağını düşünmüştüm. Bunun üzerine yazmış olduğum kod satırlarını incelemeye başladım. İlk olarak awt.window paketinde yer alan Frame sınıfından bir nesne örneği oluşturmuştum. Bunu yaparkende, yapıcı metoda string tipte bir parametre gönderdim. Bu parametre Frame penceresinin başlığı (Title) olacaktı.

```
Frame pencere=new Frame("ILK PENCEREM");
```

Daha sonra, Frame� in ekran üzerindeki konumunu belirledim. Bunun içinde setLocation metoduna X ve Y koordinatlarını 0 olarak verdim. Böylece, Frame penceresi ekranın sol üst köşesinde konumlanacaktı.

```
pencere.setLocation(0,0);
```

setBackgorund metodu ile Frame penceresinin arka plan rengini kırmızı olarak belirledim.

```
pencere.setBackground(Color.red);
```

Frame sınıfının en önemli metodu ise setVisible idi. Bu metod, oluşturulan Frame penceresinin gösterilmesini sağlıyordu. Bunun için parametre olarak metoda true değerini vermek yeterliydi.

```
pencere.setVisible(true);
```

Buraya kadar herşey sorunsuz gözüküyordu. Ancak Frame� in neden böyle göründüğünü tam olarak anlayamamıştım. Kaynaklarımı gözden geçirdiğimde, setSize isimli metodu kullanmadığımı farkettim. Bu metod ile Frame� in başlangıç boyutlarını belirleyebiliyordum. Şimdi tek yapmam gereken uygulama koduna setSize metodunu ilave etmek olacaktı. Lakin ufak bir sorun vardı. O da, Frame penceresini X butonuna basıp kapatamıyor oluşuydu. Programdan çıkamıyordum. Bunun tek bir nedeni olabilirdi o da, X butonu ile kapatma işlemi için gerekli olan olay dinleyecisinin ilgili olay metodunu çalıştırmayışıydı.

Frame sınıfının olaylarına sonradan zaten bakacaktım. Ancak bu pencereyi bir şekilde kapatıp, kodumu düzenlemek istiyordum. Yaklaşık bir yarım saat kadar sırf bu pencerenin nasıl kapatılacağını araştırdım. Nitekim windows� un ALT+F4 tuş kombinasyonu dahi işe yaramıyordu. Sonunda komut satırından CTLR+C tuş komimasyonuna basmam gerektiğini öğrendim. Bu tuş kombinasyonu sayesinde açık olan uygulama kapatılabiliyordu. Artık uygulama kodlarımı düzenleyebilir ve Frame penceresinin istediğim boyutlarda oluşturulmasını sağlayabilirdim. Bu amaçla kodlarıma aşağıdaki satırı ekledim. Burada ilk parametre Frame penceresinin genişliğini (width), ikinci parametres ise yüksekliğini (height) belirtmekteydi.

```
pencere.setSize(300,100);
```

Uygulamayı bu haliyle derleyip çalıştırdığımda 300 piksel genişliğinde ve 100 piksel yüksekliğinde bir Frame penceresi elde ettim. Artık hem Title görünüyordu, hemde Frame penceresi daha makul boyutlardaydı.



Kaynaklardan Frame ile ilgili olarak kullanabileceğim diğer teknikleride araştırmaya başladım. Örneğin, X butonunun aksine, Minimize ve Maksimize butonları çalışıyor dolayısıyla Frame penceresi mimimize edilebiliyor yada maksimize

olabiliyordu. Derken aklıma, bu Frame� in Maksimize edilmek istendiğinde, belirli yükseklik ve genişliğin üstüne çıkmamasını nasıl sağlayabileceğim sorusu geldi. Bunun için setMaximizedBounds() isimli bir metod buldum. Bu Frame sınıfına ait metoda Rectangle sınıfı türünden bir nesne parametre olarak aktarılabiliyordu. Bu Rectangle nesnesi, bir dortgen şeklini boyutları ve konumları ile bildirebildiğinden, setMaximizedBounds metodu sayesinde, Frame penceresi belirtilen Rectangle nesnesinin boyutları kadar büyüyebilecekti. Hemen bu durumu analiz etmek amacıyla uygulama kodlarını aşağıdaki gibi geliştirdim.

```
Rectangle r=new Rectangle(500,500);
pencere.setMaximizedBounds(r);
```

Burada Rectangle sınıfından nesne örneğini oluştururken, parametre olarak genişlik ve yüksekliği bildirdim. İlk parametre Rectangle nesnesinin genişliğini, ikinci parametre ise yüksekliğini belirtmekteydi. Daha sonra, setMaximizedBounds metoduna, bu Rectangle nesnesini parametre olarak verdim. Uygulamayı tekrar derleyip çalıştırdığımda ve Maksimize butonuna bastığımda, Frame� in 500 piksel X 500 piksel boyutlarına geldiğini gördüm. Normal şartlar altında bu metodu kullanmasaydım, Frame tüm ekranı kaplayacak şekilde boyutlandırılacaktı.

Frame pencereleri ile ilgili aklıma takılan bir diğer nokta ise, X butonu ile pencereyi kapatamayışımdı. Bunu kendim programlamam gerekiyordu. Bir başka deyişle, olay metodunu yazmalıydım. Kaynaklarımı araştırdığımda, Java Frame sınıfının aşağıdaki window olay metodlarına cevap verebildiğini öğrendim.

Frame için Window Olayları		
windowOpened	Pencere ilk kez gösterildiğinde çalışan olay.	WindowListener Arayüzünden
windowClosing	Pencere kullanıcı tarafından kapatılırken gerçekleşen olay.	

windowClosed	Pencere kapatıldıktan sonra çalışan olay.	
windowIconified	Pencere minimize edildiğinde gerçekleşen olay.	
windowDeiconified	Minimize olan bir Pencere normal haline döndüğünde gerçekleşen olay.	
windowActivated	Pencereya odaklanıldığı (Focus) yani aktifleştirildiği zaman çalışan olay.	
windowDeactivated	Pencereden ayrılındığında çalışan olay.	
windowLostFocus	Focus (odak) pencereden uzaklaştığında çalışan olay.	WindowFocusLis Arayüzünden
windowGainedFocus	Odak (Focus) pencereye geldiğinde çalışan olay.	
windowStateChanged	Pencerenin durumu değiştiğinde (minimize edildiğinde, maksimize edildiğinde vb.) çalışan olay.	WindowStateList Arayüzünden

İlk olarak denemek istediğim, pencerenin X butonu ile kapatılabilmesiydi. Öncelikle, windowClosing metodunu uygulamam gerekiyordu. Bunu gerçekleştirebilmek için, WindowListener arayüzünü sınıfa uygulamalıydım. Böylece, WindowListener arayüzünden uyguladığım windowClosing metodunda yazabilir ve X butonu ile pencerenin kapatılaması sırasında oluşacak olayı kodlayabilirdim. Bu amaçla sınıf kodlarını aşağıdaki gibi geliştirdim.

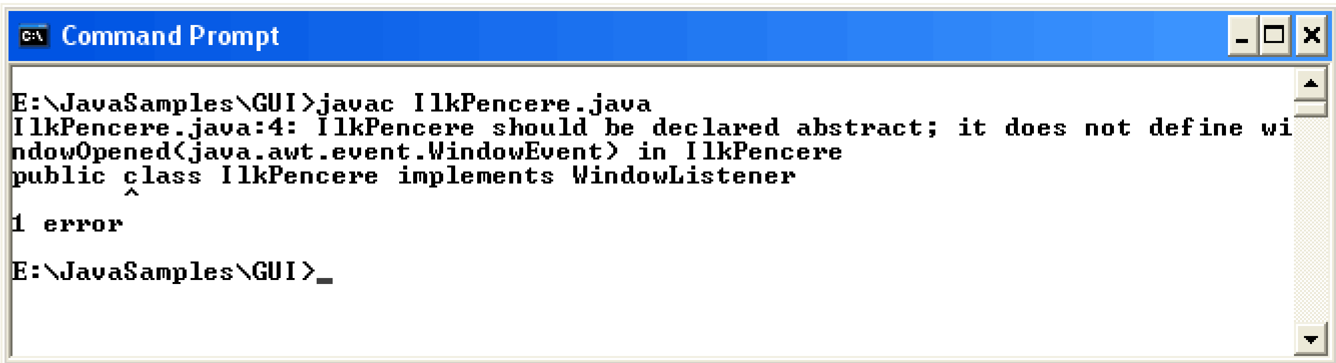
```
import java.awt.*;
import java.awt.event.*;

public class IlkPencere implements WindowListener
{
    public static void main(String args[])
    {
```

```
        IlkPencere p=new IlkPencere();
        Frame pencere=new Frame("ILK PENCEREM");
        pencere.setLocation(0,0);
        pencere.setBackground(Color.red);
        pencere.setSize(300,100);
        Rectangle r=new Rectangle(500,500);
        pencere.setMaximizedBounds(r);
        pencere.addWindowListener(p);
        pencere.setVisible(true);
    }

    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```

Programı bu haliyle derlediğimde aşağıdaki hata mesajını aldım.



Anladığım kadarı ile WindowListener arayüzündeki tüm window olay metodlarını sınıf içerisinde kullanmasamda bildirmeliydim. Bu amaçla sınıfa aşağıdaki metodlarıda ekledim.

```
public void windowOpened(WindowEvent e)
{
}
public void windowClosed(WindowEvent e)
{
}
public void windowIconified(WindowEvent e)
{
```

```
}
public void windowDeiconified(WindowEvent e)
{
}
public void windowActivated(WindowEvent e)
{
}
public void windowDeactivated(WindowEvent e)
{
}
```

Uygulama başarılı bir şekilde derlendikten sonra, hemen X butonu ile kapatılıp kapatılamadığını denemedim. Sonuç başarılıydı. Elbetteki bir pencere bu haliyle çok yavan durmaktaydı. Bu pencereye kontroller eklemek gerekiyordu. Normal bir Applet� e kontroller nasıl ekleniyorsa buradada aynı kurallar geçerliydi. Bu kez bir Applet� e kontrol eklemek yerine bir Frame nesnesine kontrol ekleyecektim. Bu amaçla uygulamayı biraz daha düzenlemeye ve ilginç hale getirmeye karar verdim. Amacım Frame içindeki bir button yardımıyla başka bir frame penceresinin açılabilmesini sağlamaktı. Bu amaçla aşağıdaki örneği oluşturdum.

```
import java.awt.*;
import java.awt.event.*;

public class IlkPencere implements
WindowListener,ActionListener
{
   public Frame p1;
   public Button btnIkinciPencere;
   public Button btnKapat;
   public int X;
   public int Y;

   public void PencereAyarla(String baslik,int genislik,int
yukseklik,int konumX, int konumY,Color
arkaPlanrengi)
   {
      X=konumX;
```

```
      Y=konumY;

      p1=new Frame(baslik);
      p1.setLocation(konumX,konumY);
      p1.setBackground(arkaPlanrengi);
      p1.setSize(genislik,yukseklik);
      p1.setLayout(new FlowLayout());
      p1.addWindowListener(this);

      btnIkinciPencere= new Button("Ikinci Pencere");
      btnKapat=new Button("Kapat");

      p1.add(btnIkinciPencere);
      p1.add(btnKapat);

      btnKapat.addActionListener(this);
      btnIkinciPencere.addActionListener(this);

      p1.setVisible(true);
   }

   public static void main(String args[])
   {
      IlkPencere p=new IlkPencere();
      p.PencereAyarla("ANA
PENCERE",250,100,0,0,Color.white);
   }

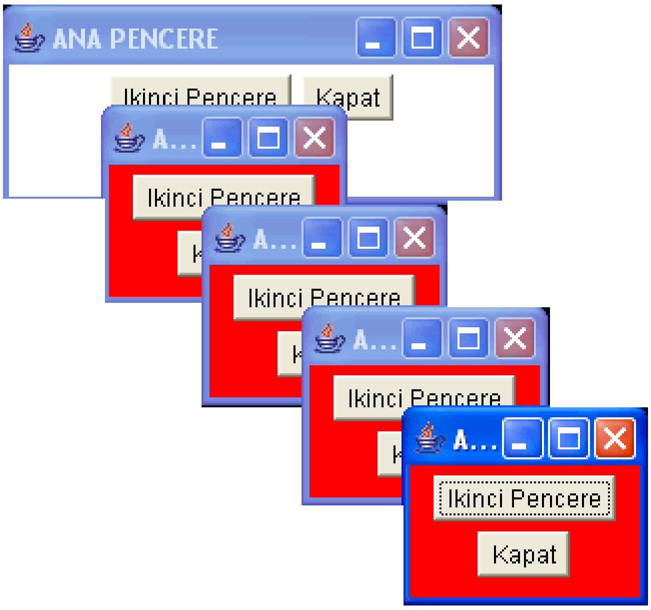
   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==btnKapat)
      {
         p1.setVisible(false);
      }
      else if(e.getSource()==btnIkinciPencere)
      {
         IlkPencere p=new IlkPencere();
         X=X+50;
         Y=Y+50;
         p.PencereAyarla("ANA
PENCERE",100,100,X,Y,Color.red);
```

```
        }
    }
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
    public void windowOpened(WindowEvent e)
    {
    }
    public void windowClosed(WindowEvent e)
    {
    }
    public void windowIconified(WindowEvent e)
    {
    }
    public void windowDeiconified(WindowEvent e)
    {
    }
    public void windowActivated(WindowEvent e)
    {
    }
    public void windowDeactivated(WindowEvent e)
    {
    }
}
```

Bu uzayıp giden kodlar çok işe yaramıyor. Ancak şu ana kadar GUI �ler ile ilgili bilgilerimi tekrar etmemede yardımcı oldu. Bu uygulama çalıştığında ilk olarak belirtilen boyutlarda, konumda, başlıkta ve art alan renginde bir ana pencere oluşturuyor. Bu pencere üzerine, FlowLayout sınıfının öngördüğü Layout düzenine göre yerleşen iki Button bileşenim var. Ikınci Pencere başlıklı button bileşenine tıklandığında yeni bir pencere oluşturuluyor. Kapat button bileşeni ise, bu pencereyi kapatıyor. Bu kapatma işleminde setVisible(false) metodunu kullandım. Böylece sonradan açılan pencereler aslında gizleniyordu.

Uygulamayı bu haliyle derleyip çalıştırdığımda aşağıdaki gibi bir görüntü oluştu. Her yeni pencere bir öncekinin konumunun 50 birim sağına ve altına konumlandırılıyor. Elbette X butonuna

basıldığında System.exit(0) metodu o an çalışan prosesi sonlandırdığı için tüm pencereler kapanmaktaydı. Mesela ilk pencerede Kapat başlıklı butona basınca komut satırı açık kalacak şekilde pencere ortadan kayboluyor. Yani görünmez oluyor. Ancak proses çalışmaya devam ediyor. Sanırım neden işe yaramaz bir program olduğu ortada. Olsun en azından el cimnastipi yapmış oldum.



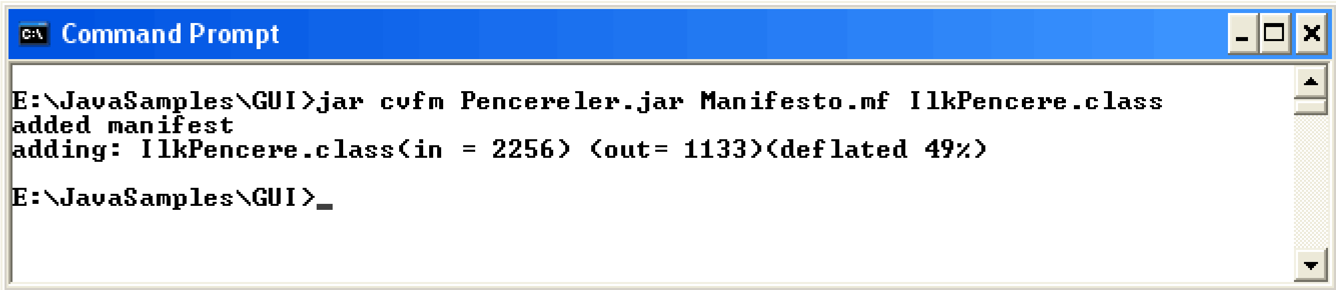
Şu anada kadar yaptıklarım ile geliştirdiğim bu pencere uygulamalarında önemli bir sorun var aslında. Bu uygulamaları çalıştırabilmek için komut satırında ilgili sınıfı java yorumlayıcısı ile açmam gerekiyor. Diğer taraftan uygulama çalışırken, komut satırı açık kalıyor. Oysaki normal bir exe dosyası gibi bu uygulamanın tek başına çalışabilmesi çok daha yerinde olur. İşte bunu gerçekleştirmek için kaynaklarda 3 yoldan bahsedildiğini öğrendim. En basit olanı üçüncü parti yazılımlar ile bu işi gerçekleştirmek. Örneğin halen daha özlemini çektiğim değerli arkadaşımın bilgisayarında yer alan JBuilder gibi.

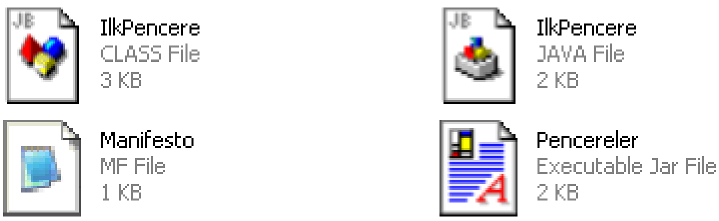
Diğer iki yol ise bizim manuel olarak kullanabileceğimiz teknikler içeriyor. Bunlardan birisi Dos ortamından kalma bat(batch) uzantılı dosyalar içerisine uygulamayı çalıştıracak kod satırını yazmak. Diğeri ise, GUI uygulamasına ati tüm sınıfları ve gerekli dosyaları içeren bir JAR paketi oluşturmak. Açıkçası JAR paketini oluşturmak bana daha mantıklı göründü. Ancak bir JAR paketini oluşturmadan önce, bu JAR paketi için verisyon numarası, ana sınıf gibi bilgileri içeren bir manifesto

dosyası hazırmamam gerektiğini öğrendim. Bu manifesto dosyası, mf uzantılı olmakla birlikte, aslında .net assembly� larındaki manifesto bilgilerinin tutulduğu yapıya benzer bir içeriğe sahip. Çok basit olarak geliştridiğim java uygulaması için aşağıdaki bilgileri içeren bir manifesto dökümanı hazırladım.

```
Manifest-Verison: 1.0
Main-Class: IlkPencere
Created-By: 1.4.1 (Sun Microsystems Inc.)
```

Bu dosyayı Manifesto.mf ile kaydettikten sonra aşağıdaki komut ile, Jar dosyasını oluşturdum.





Artık Pencereler.Jar dosyasına çift tıkladığımda GUI uygulamasının, normal bir windows uygulaması gibi çalıştığını gördüm. Bu sorunu çözmem son derece önemli idi. Artık windows tabanlı GUI� lerin nasıl oluşturulduğunu, window olaylarına nasıl cevap verdiğini biliyordum. Dahası bu pencereler üzerine awt bileşenlerinin nasıl ekleneceğini ve herşeyden önemlisi bu GUI uygulamasının çift tıklamalı versiyonunun Jar dosyası olarak nasıl hazırlanabileceğini biliyordum. Artık tüm bu bildiklerimi birleştirerek daha işe yarar bir uygulama yapabileceğim kanısındaydım. Bunun için aklıma basit bir hesap makinesi uygulması yazmak geldi. Ama çok basit. Sadece 2 operand değeri için 4 işlem yapacaktı. Lakin burada önemli olan, Frame� in tasarlanması ve Frame üzerindeki bileşenlerin olaylara tepki vermesinin sağlanmasıydı.

Hemen kolları sıvadım ve uygulamayı geliştirmeye başladım. Sonuçta hem pratik yapmış oldum hemde GUI bilgilerimi tekrar etmiş. Sonuçta aşağıdaki küçük programcık ortaya çıktı.

```
import java.awt.*;
import java.awt.event.*;

/* HesapMakinesi sınıfında window olaylarına ve Button
olaylarına izin verebilmek için, WindowListener ve
ActionListener arayüzlerinin uygulanması gerekir. */
public class HesapMakinesi implements
WindowListener,ActionListener
{
    /* Frame sınıfına ait nesne tanımlanıyor ve bu Frame
üzerindeki awt bileşenleri tanımlanıyor.*/
    public Frame f;
    public Button btnHesapla;
    public Label lbSayi1;
    public Label lbSayi2;
    public Label lbIslem;
    public TextField tfSayi1;
    public TextField tfSayi2;
    public Choice lstIslem;

    /* iki sayı değerini ve işlem sonucunu tutacak double
tipinden değişkenler tanımlanıyor.*/
    public double sayi1,sayi2,sonuc;

    /* Olustur metodunda, penceremiz ve üzerindeki bileşenler
oluşturuluyor.*/
    public void Olustur()
    {
        f=new Frame("Hesap Makinesi"); // Başlığı (Title) Hesap
Makinesi olan bir Frame nesnesi oluşturuluyor.
        f.setLayout(new FlowLayout()); // Frame üzerindeki
bileşenler FlowLayout tekniğine göre dizilecekler.
        Color c=new Color(248,221,139); /* Color tipinden bir
nesne R (Red), G (Green), B(Blue) formatında oluşturuluyor.*/
        f.setBackground(c); // Pencerenin arka plan rengi c isimli
Color nesnesine göre belirleniyor.
```

```
        /* TextField bileşenleri 10 karakter uzunluğunda
oluşturuluyor.*/
        tfSayi1=new TextField(10);
        tfSayi2=new TextField(10);

        /* Label bileşenleri başlıkları ile oluşturuluyor.*/
        lbSayi1=new Label("Sayi 1");
        lbSayi2=new Label("Sayi 2");
        lbIslem=new Label("ISLEMIN SONUCU...");

        /* Button bileşeni oluşturuluyor ve bu bileşen için olay
dinleyicisi ekleniyor.*/
        btnHesapla=new Button("Hesapla");
        btnHesapla.addActionListener(this);

        /* Choice (başka bir deyişle ComboBox) bileşeni
oluşturuluyor. Listedeki elemanlar addItem metodu ile
ekleniyor.*/
        lstIslem=new Choice();
        lstIslem.addItem("TOPLA");
        lstIslem.addItem("CIKART");
        lstIslem.addItem("BOL");
        lstIslem.addItem("CARP");

        /* Bileşenler sırasıyla Frame bileşenine yani pencereye
add metodu ile ekleniyor. */
        f.add(lbSayi1);
        f.add(tfSayi1);
        f.add(lstIslem);
        f.add(lbSayi2);
        f.add(tfSayi2);
        f.add(btnHesapla);
        f.add(lbIslem);
        f.pack(); /* pack metodu ile pencerenin yüksekliği ve
genişliği, içerdiği bileşenlerin kapladığı alana göre otomatik
olarak ayarlanıyor.*/
        f.addWindowListener(this); // Frame bileşeni için window
olay dinleyicisi ekleniyor.
```

```
        f.setVisible(true); // Frame bileşeni (pencere) gösteriliyor.
    }

    /* IslemYap metodunda 4 işlem gerçekleştiriliyor. */
    public void IslemYap()
    {
        sayi1=Double.parseDouble(tfSayi1.getText()); /*
TextField bileşenlerinin string içeriği Double sınıfının
parseDouble metodu ile double tipine dönüştürülerek değişkene
atanıyor.*/
        sayi2=Double.parseDouble(tfSayi2.getText());

        /* if koşullarında Choice bileşeninde seçili olan item
getSelectedItem() metodu ile alınıyor ve uygun olan işlemler
yapılıyor.*/
        if(lstIslem.getSelectedItem()=="TOPLA")
        {
            sonuc=sayi1+sayi2;
            bIslem.setText(sayi1+"+"+sayi2+"="+sonuc);
        }
        else if(lstIslem.getSelectedItem()=="CARP")
        {
            sonuc=sayi1*sayi2;
            lbIslem.setText(sayi1+"x"+sayi2+"="+sonuc);
        }
        else if(lstIslem.getSelectedItem()=="CIKART")
        {
            sonuc=sayi1-sayi2;
            lbIslem.setText(sayi1+"-"+sayi2+"="+sonuc);
        }
        else if(lstIslem.getSelectedItem()=="BOL")
        {
            sonuc=sayi1/sayi2;
            lbIslem.setText(sayi1+"/"+sayi2+"="+sonuc);
        }
    }

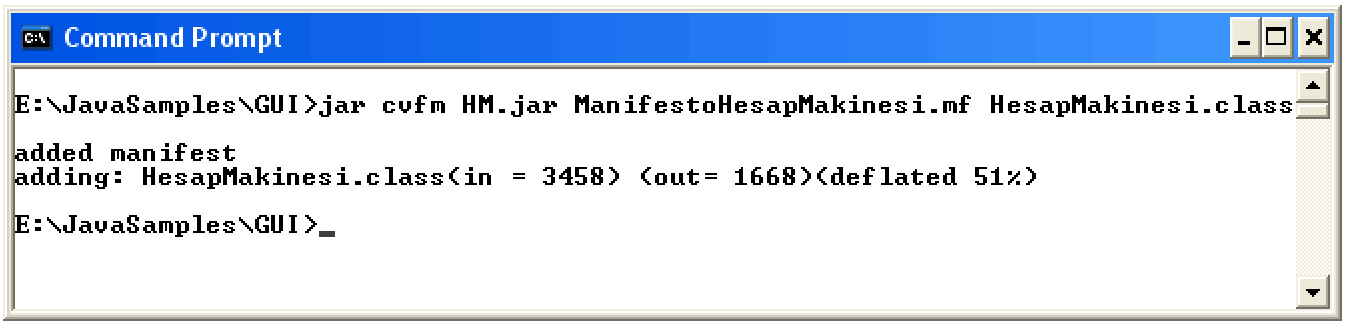
    public static void main(String args[])
    {
        HesapMakinesi m=new HesapMakinesi();
```

```
        m.Olustur();
    }
    /* actionPerformed olayı meydana geldiğinde bu metod
çalışıyor.*/
    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource()==btnHesapla) // Eğer olayın kaynağı
Button bileşeni ise, yani Button� a tıklandıysa.
        {
            IslemYap();
        }
    }

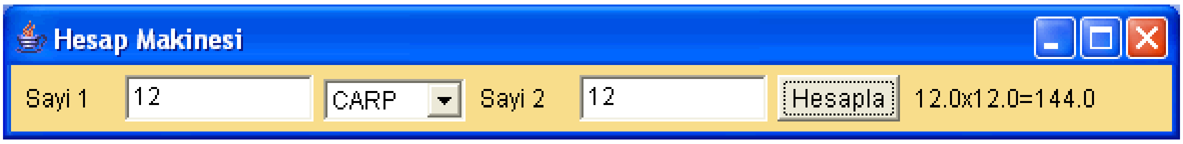
    /* Kullanıcı X buton ile pencereyi kapatmak istediğinde bu
olay metodu çalışıyor. */
    public void windowClosing(WindowEvent e)
    {
        System.exit(0); /* Güncel olan proses sonlandırılıyor.
Dolayısıyla uygulama sona eriyor. */
    }
    public void windowOpened(WindowEvent e)
    {
    }
    public void windowClosed(WindowEvent e)
    {
    }
    public void windowIconified(WindowEvent e)
    {
    }
    public void windowDeiconified(WindowEvent e)
    {
    }
    public void windowActivated(WindowEvent e)
    {
    }
    public void windowDeactivated(WindowEvent e)
    {
    }
}
```

Programı derledikten sonra Manifesto dosyasını (ManifestoHesapMakinesi.mf) aşağıdaki gibi düzenledikten sonra, JAR Paketinide hazırladım.

```
Manifest-Verison: 1.0
Main-Class: HesapMakinesi
Created-By: 1.4.1 (Sun Microsystems Inc.)
```



Paketi çift tıkladığımda, basit hesap makinesi uygulamam kullanılmaya hazırdı.



Artık GUI� lerde iyice ilerlemeye başladığımı hissediyordum. Bununla birlikte, 2 boyutlu grafik çizimleri, animasyon hazırlamak, resim işlemek, ses işlemek, Swing bileşenleri, Menu� ler vs... gibi henüz bilmediğim daha pek çok konu vardı. Ancak hem kahvem hemde pilim bitmişti. Sanırım önümüzdeki günlerde, bu konulara eğileceğim.

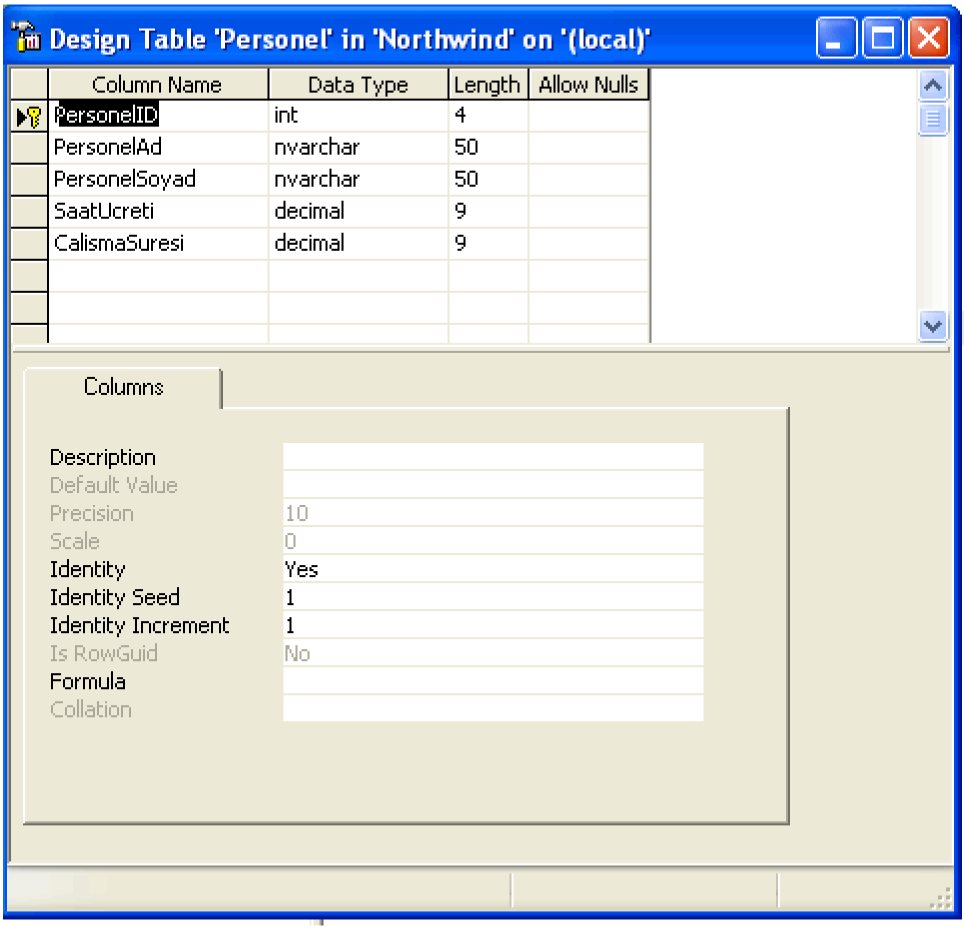
Bölüm 19: JBuilder ile Database Uygulamaları

Bu hafta oldukça mutluyum. Nitekim, JBuilder programını kullanan arkadaşım tatilden döndü. İlk fırsatta ona uğramayı ve JBuilder ile yeni bir şeyler yapmayı düşünüyordum. Ofisten içeri girdiğimde, arkadaşım beni görünce hafif bir tebessüm ile şöyle dedi; "Bir kahve molası veriyoruz ha!". Başımı hafifçe öne eğmem, ofise geliş amacımı açıkça ortaya koymuştu. Kısa bir

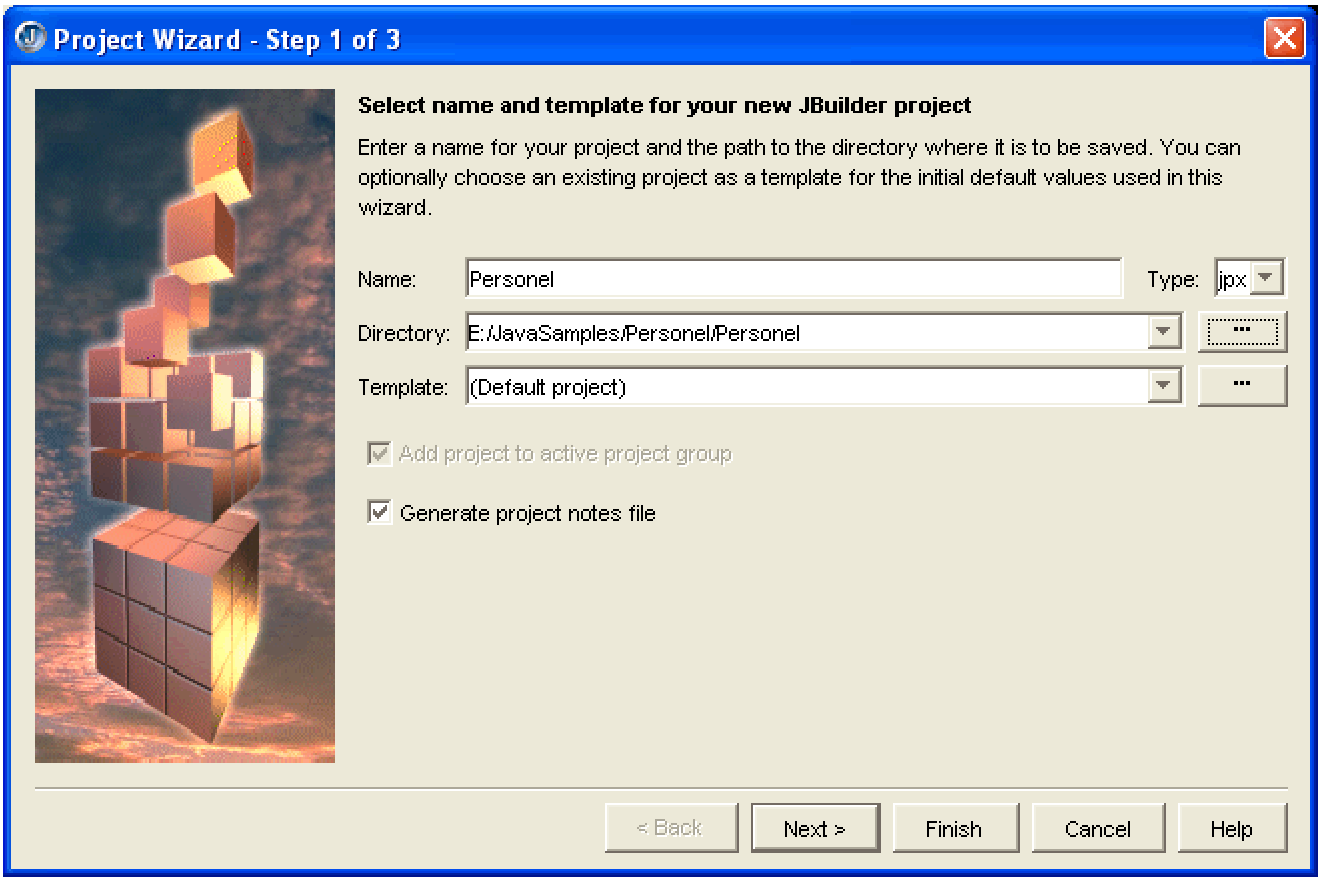
muhabbetin ve güzel yaz tatili serüvenlerinin ardından, arkadaşımın bilgisayarı ve JBuilder ile baş başa kalmıştım. Uzun süredir merak ettiğim bir uygulamayı geliştirmek istiyordum. Bir veritabanı uygulaması. Açıkçası JBuilder ile bu tarz bir uygulamanın nasıl yazılacağını merak ediyordum.

Bundan bir kaç sene öncesine kadar bir Delphi programcısı olmam, Borland firmasının yazılım geliştirme uygulamalarına olan aşinalığımı arttıran bir etken olmuştu. Delphi programlama dilinin bana verdiği en önemli kazanç, gelişmiş bileşen paketleri sayesinde, patronun benden istediği uygulamaları zamanından çok önce geliştirebilmem olmuştur. JBuilder uygulama geliştirme ortamı içinde aynı şeyin söz konusu olduğundan emindim. Nitekim bu bir yandan iyi, diğer yandanda kötü bir olgudur. Delphi gibi geliştirme ortamları, bir programcının analitik düşünce sistemini ve kod yazma alışkanlıklarını tembelleştirir. Zaten bu bu .net platformuna geçip C# dilini öğrenmemin nedenlerinden sadece birisiydi. Ancak, yinede zaman zaman uygulama geliştirirken Visual Studio.Net ortamını kullanmakta ve avantajlarından yararlanmaktayım. Buna rağmen, Visual Studio.Net, programcıyı daha çok kod yazmaya ve algoritma geliştirmeye davet ederek daha bilgili olmamızı gerektiriyor diyebilirim. Sonuç olarak yinede, JBuilder gibi bir geliştirme ortamını kullanarak neler yapılabileceğinide görmekte fayda var. Özellikede yoğun projelerde çalışanlar için, uygulama geliştirme süresinin kısaltılması son derece önemli bir faktör.

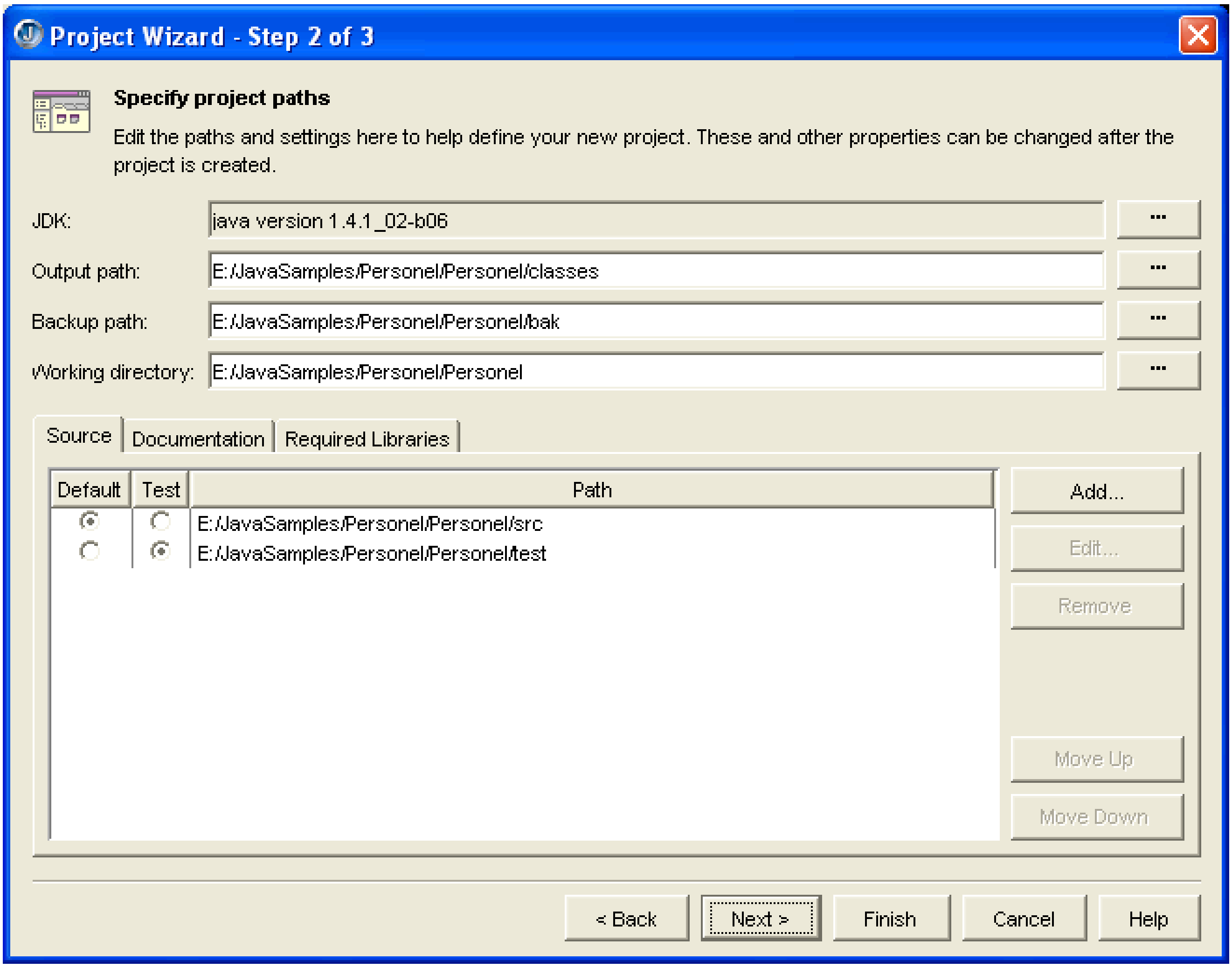
Ben bu düşüncelerim ile boğuşurken bir anda kendimi, JBuilder ile veritabanı uygulaması geliştirirken bulmuştum. Öncelikle, işin nasıl yapıldığını kavramam gerekiyordu. Bu amaçla JBuilder' ın tutorial' larında biraz inceleme yaptım. Daha sonra ise uygulamayı geliştirmeye başladım. Herşeyden önce bana, kullanabileceğim bir veritabanı gerekliydi. Bunu gerçekleştirebilmek için, yerel sql sunucusu üzerinde yer alan Northwind veritabani altında personel isimli bir tablo oluşturdum.



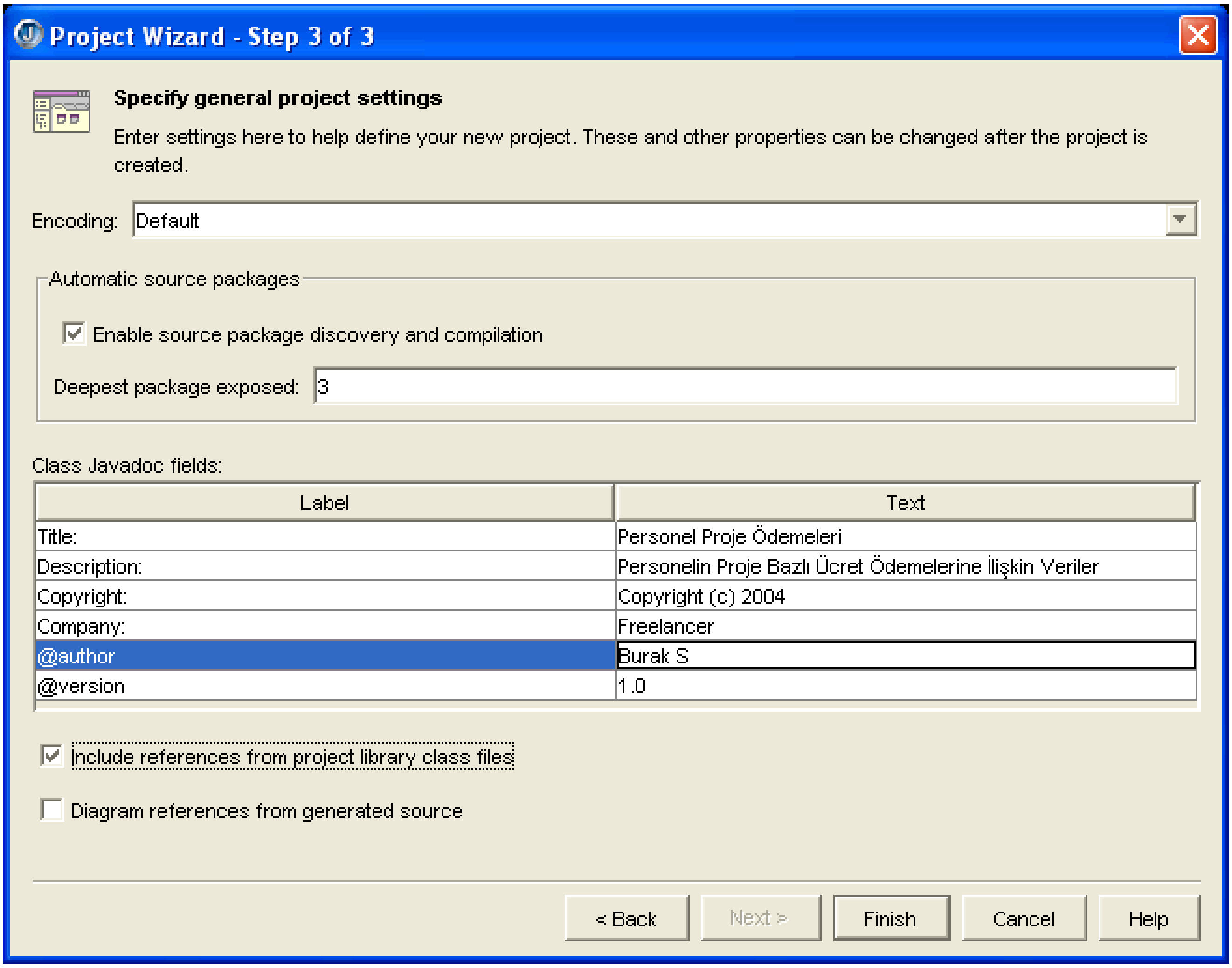
Personel tablosunda, personelin saat ücreti ve çalıştığı süre ile isim ve soyisim bilgileri yer almakta. Aslında bu tabloda toplam ücretin yer aldığı bir sütunda oluşturulabilirdi. Ancak ben bu sütunu, JBuilder uygulaması içinde bir calculated field olarak oluşturmayı planlıyorum. Personel tablosunu bu şekilde oluşturduktan sonra, bir kaç satırda veri girdim. Artık JBuilder uygulamasını geliştirmeye başlayabilirdim. İlk yapmam gereken, bir Java Projesi açmaktı. Projemi sihirbaz adımlarını izleyerekten kısa bir sürede oluşturdum. Önce birinci adım,



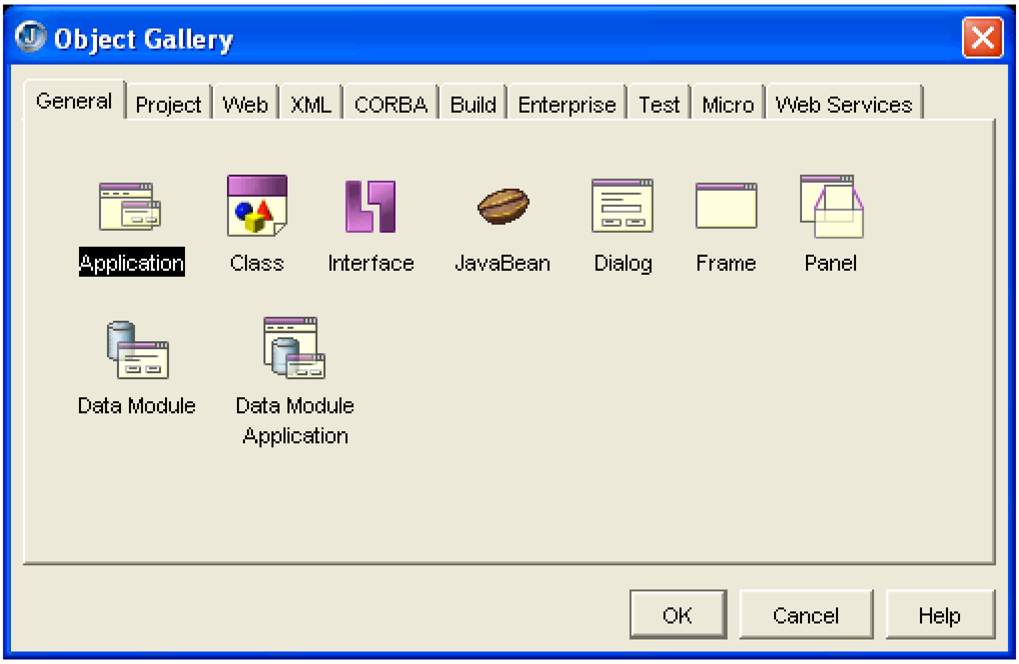
Sonra ikinci adım,



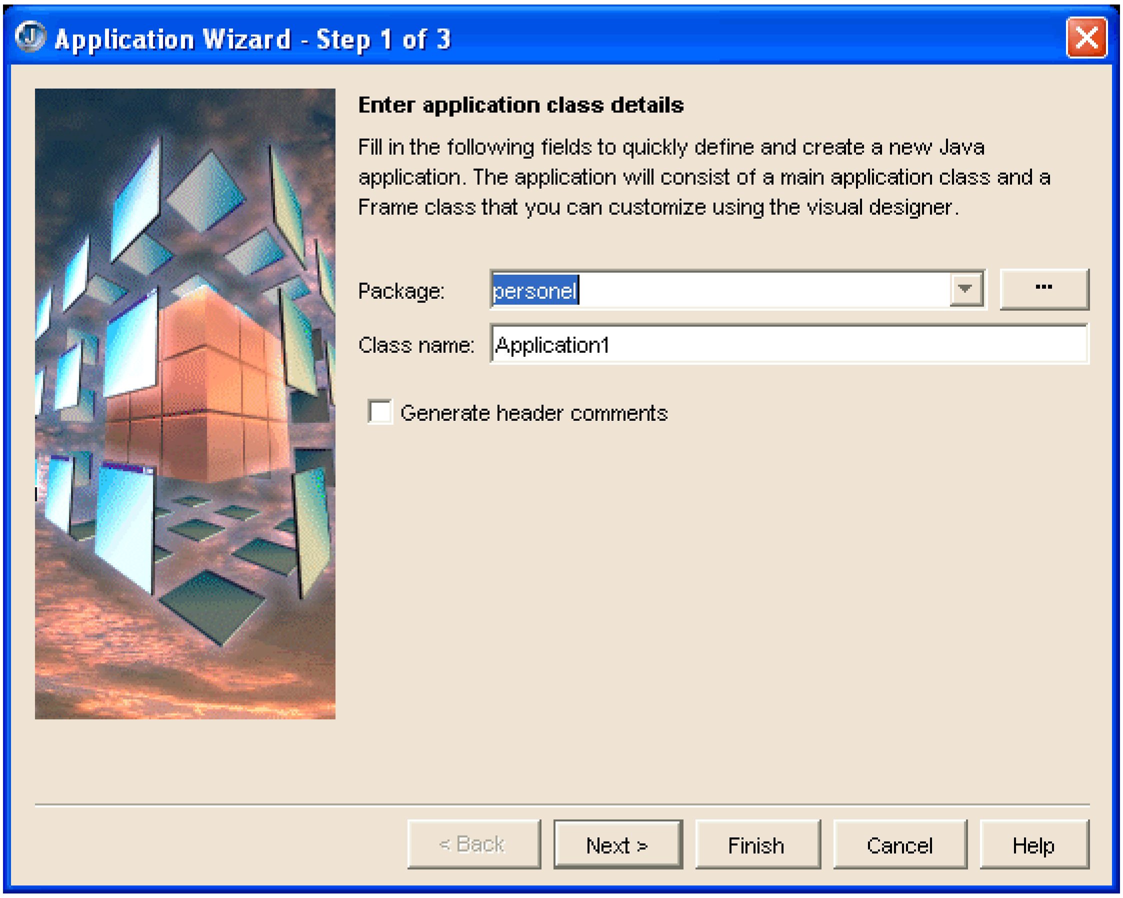
Son olarakta üçüncü adım,



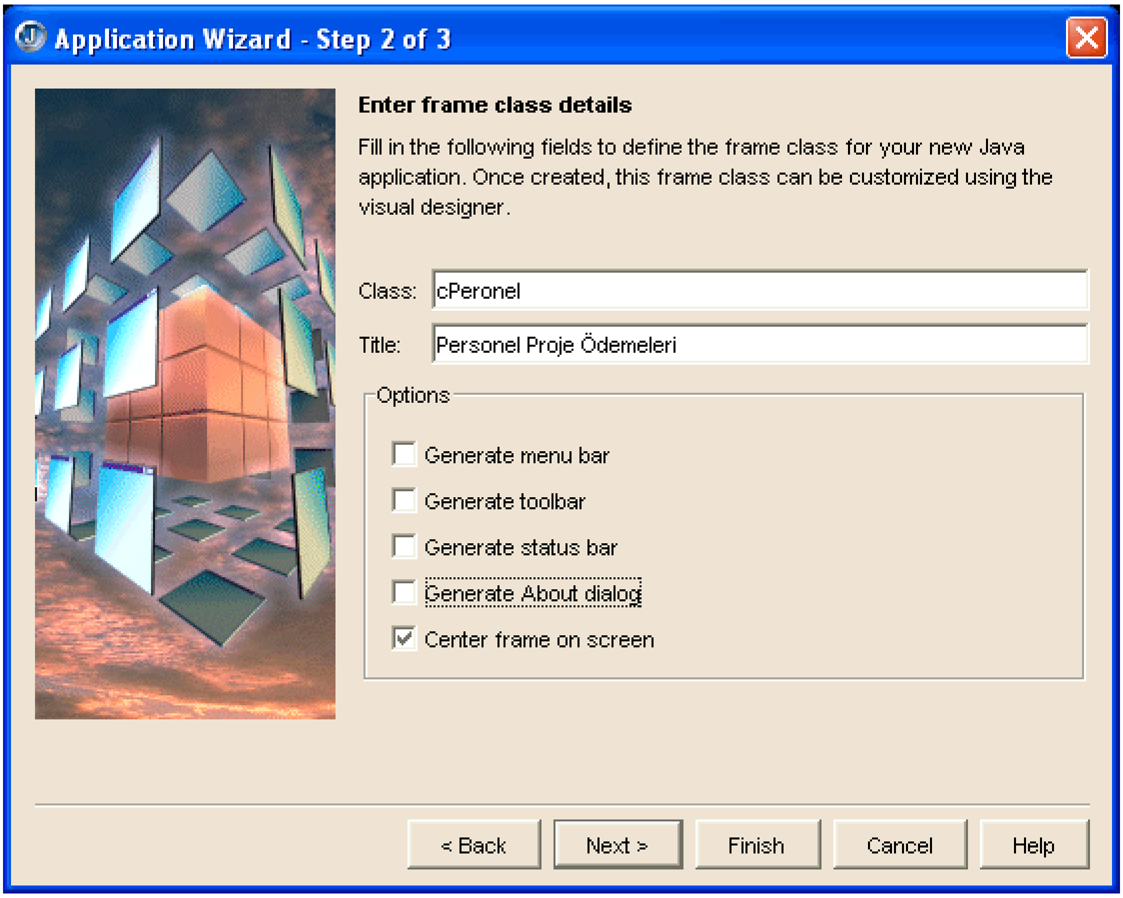
Buraya kadar yapılan işlemler, JBuilder ile oluşturduğum Applet uygulamasındaki adımlar ile aynıydı. Sırada, uygulama tipini seçmem ve oluşturmam vardı. Bu sefer, windows üzerinde çalışacak bir uygulama geliştirmek istiyordum. Bu nedenle, File->New menüsünden açılan Object Gallery iletişim penceresindeki General kısmından Application şablonunu seçtim.



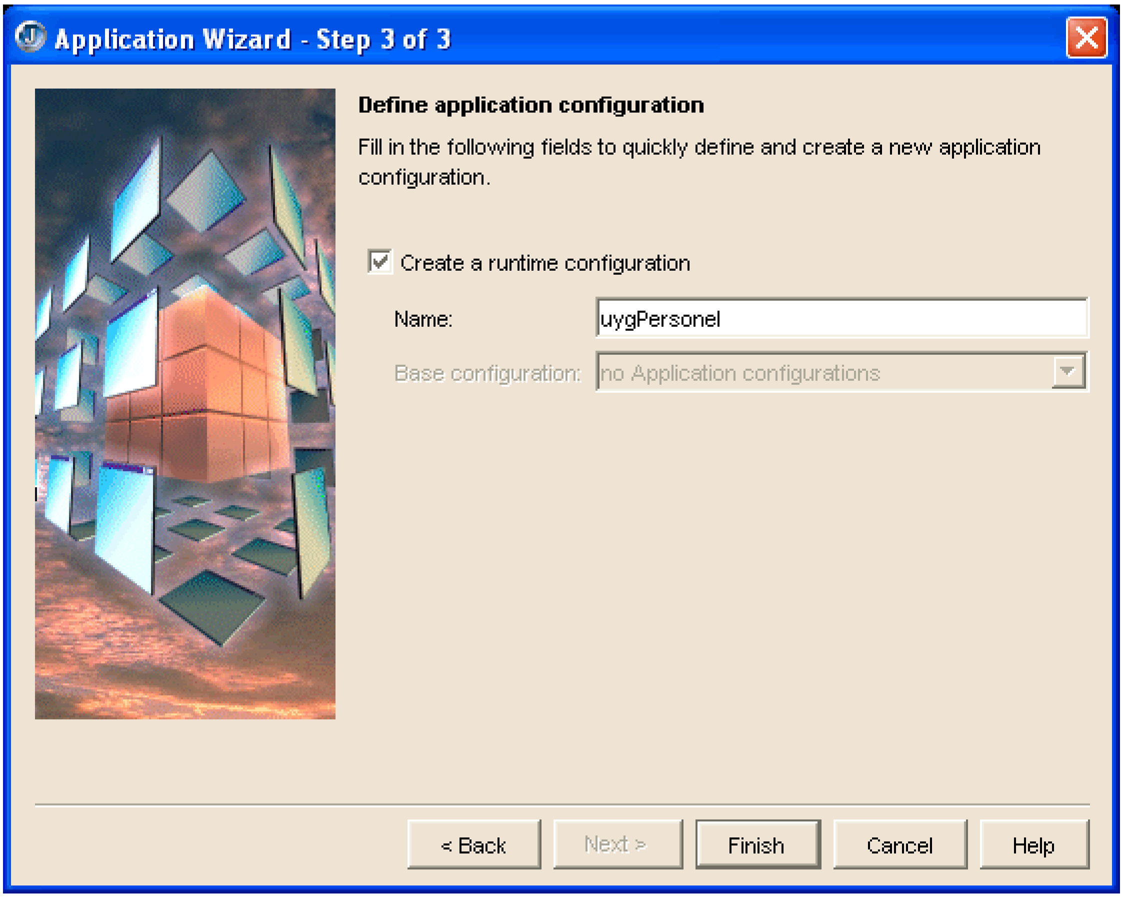
Elbette seçtiğim Application şablonu için belli adımları işlemem gerekiyordu. Karşımdaki sihirbaz adımlarınıda hızlı bir şekilde gerçekleştirerek yoluma devam ettim. Önce ilk adım,



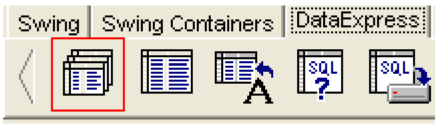
Arından ikinci adım,



Ardından üçüncü ve son adım.

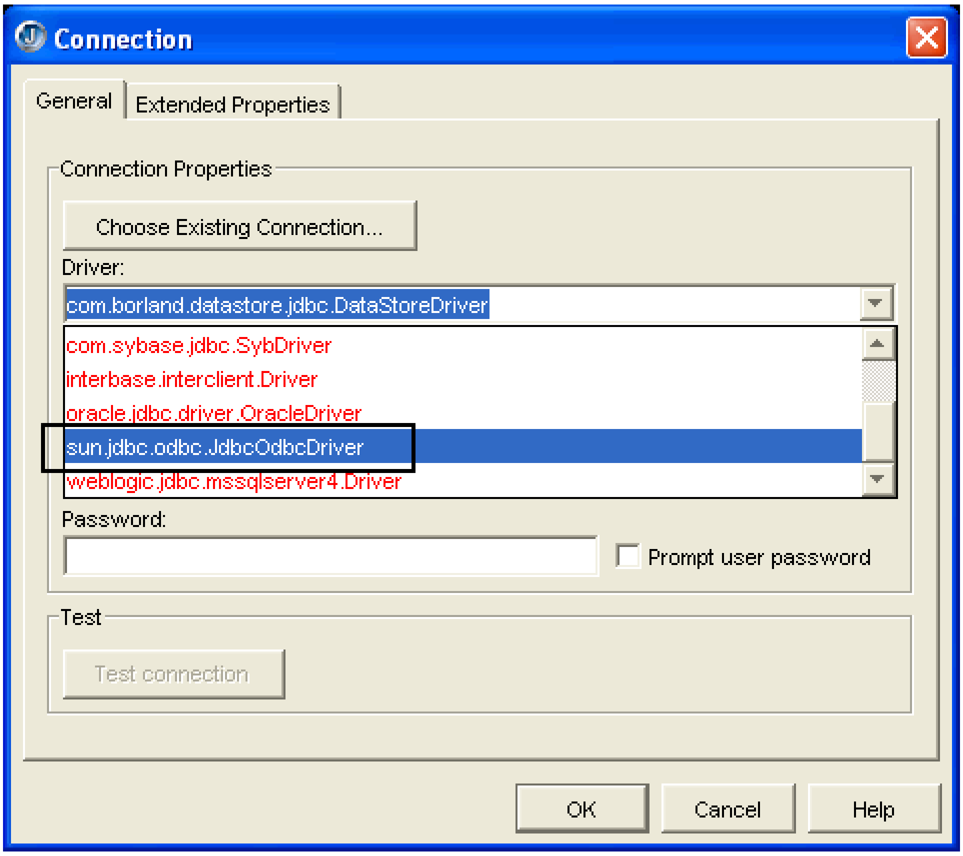


Application için gerekli Frame' de oluşturulduktan sonra, ilk yaptığım iş, Frame' in Layout özelliğini XYLayout olarak ayarlamak oldu. Nedense bu Layout' lardan çok çektim Java' da. Ancak JBuilder benim için esnek bir çözüm sağlamıştı. XYLayout ile, bileşenlerimi frame üzerine bağımsız bir şekilde yerleştirebilmekteydim. Bu adımlara rağmen halen daha veritabanı uygulamasını yazmaya başlayamamıştım. İlk olarak neye ihtiyacım olduğunu düşündüm. Net ortamında uygulama geliştirirken, veritabanına doğru bir bağlantı hattını tesis etmeme yardımcı olacak Connection nesnelerini kullanıyordum. İlk önce bileşenler paletinde, bu tarz bir öğe aradım. DataExpress sekmesindeki Database bileşeni bu arayışımın cevabı oldu. Bu bileşen ile, bağlanmak istediğim veritabanına bir hat çekebileceğimi düşünüyordum.

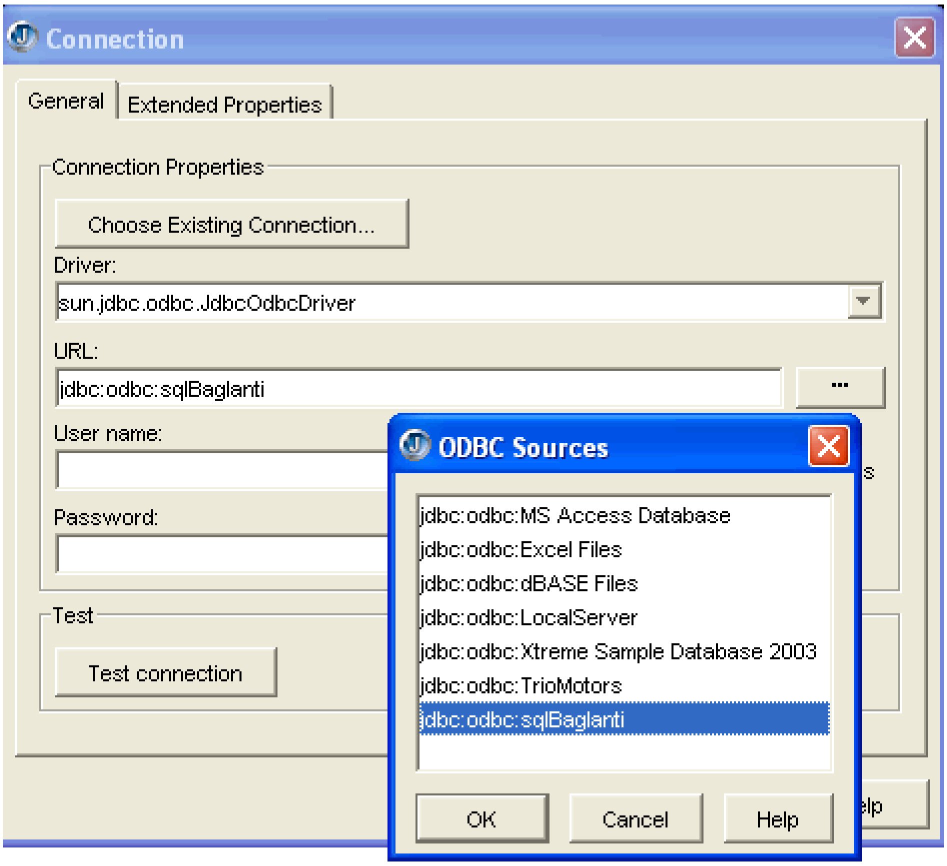


Hemen Database bileşenini Frame üzerine çizdim ve özelliklerine baktım. Connection özelliğine tıkladığımda, bağlantı

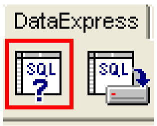
ayarlarını yapabileceğim, Connection iletişim penceresi ile karşılaştım. Bu iletişim penceresinde yer alan Driver kısmında, sistemdeki veritabanlarına erişimimi sağlayacak çeşitli java veri sürücülerinin olduğunu farkettim. Ancak bunlardan bazıları kırmızı font ile yazılmışlardı. Sonradan, kırmızı font ile yazılmış veri sürücülerinin sistemde yüklü olmadığını anladım. Elimde, kullanabileceğim JdbcOdbcDriver sürücüsü vardı. Bu sürücü ile sistemdeki ODBC kaynaklarına bağlanabilirdim.



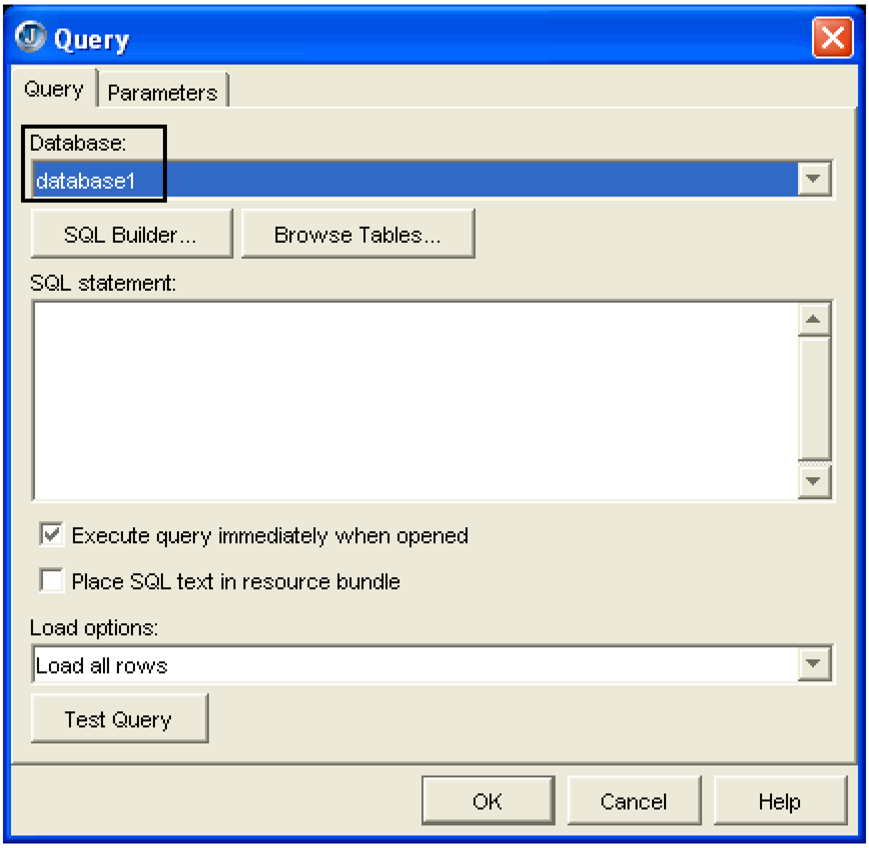
Bu sürücüyü seçtikten sonra ise, URL kısmına girdim. Ancak tabiki, sql sunucusunda yer alan Northwind veritabanı için bir System DSN' i tanımlamadığımdan, Northwind veritabanım bu listede görünmüyordu. Hemen Administrative Tool kısmından, ODBC ayarlarına girdim ve sqlBaglanti isimli bir System DSN' ini oluşturdum. Tekrar, JBuilder ortamına döndüğümde, URL sekmesinde, henüz oluşturduğum System DSN' inin eklendiğini farkettim.



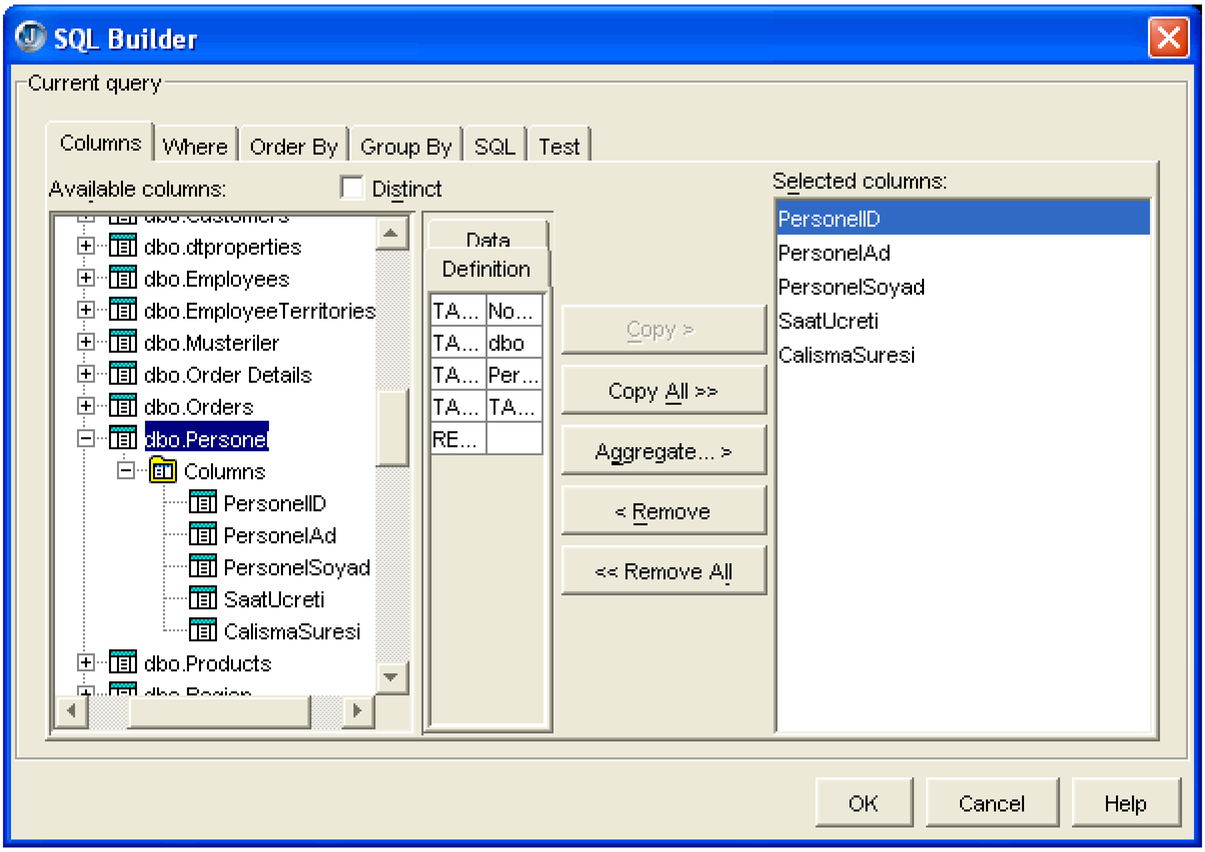
Bu işlemlerin ardından Test Connection butonua basarak bağlantının başarılı bir şekilde sağlanıp sağlanmadığına baktım. Artık veritabanıma üstelikte yerel makinedeki sql sunucusuna bağlanabileceğim bir Database bileşenim olmuştu. Şimdi ise, bu veritabanındaki Personel isimli tabloda yer alan verileri bir şekilde uygulama ortamına çekmem gerekiyordu. Bunun için bir sql query' si kullanabilirdim. Ancak bu sql query' sini çalıştıracak ve bir veri kümesi olarak ortama aktaracak bir bileşene ihtiyacım vardı. Kaynaklarımdan yaptığım incelemelerden sonra, DataExpress sekmesinde yer alan QueryDataSet bileşeninin aradığım kontrol olduğunu öğrendim.



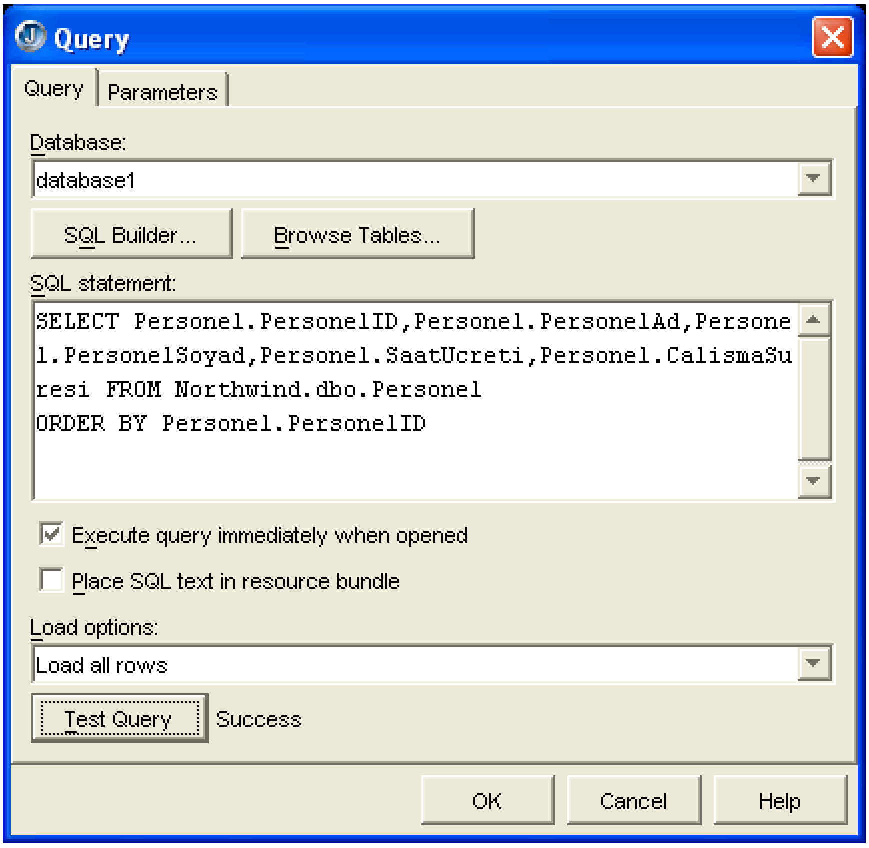
Bu bileşenin en önemli özelliği, query özelliğiydi. Query özelliğine girdiğimde, Query iletişim penceresi ile karşılaştım. Bu iletişim penceresinde, kullanacağım bağlantıyı sağlayan database bileşenini seçtim. Böylece query bileşenime, hangi veritabanı üzerindeki tablolar ile çalışacağını belirtmiş oluyordum.



Query bileşeninin, kullanacağı sorguyu, SQL Builder sekmesinden seçebileceğim gibi, direkt olarak SQL statement kısmınada yazabilirdim. SQL Builder kısmı, sql sorgusunu kolayca ve detaylı bir şekilde hazırlayabileceğim bir seçenek sunuyordu. Burada filtreleme, gruplama, sıralama ve sorgunun testi gibi işlemleri kolayaca gerçekleştirebileceğim adımlar yer almaktaydı.

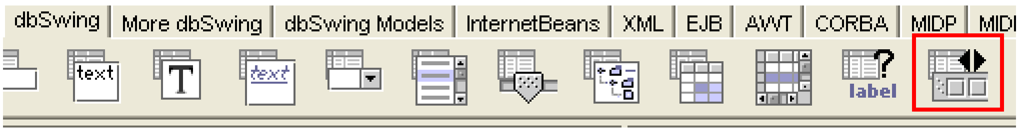


Sonuç olarak burada oluşturduğum sql cümleciği aşağıdaki gibi olmuştu.

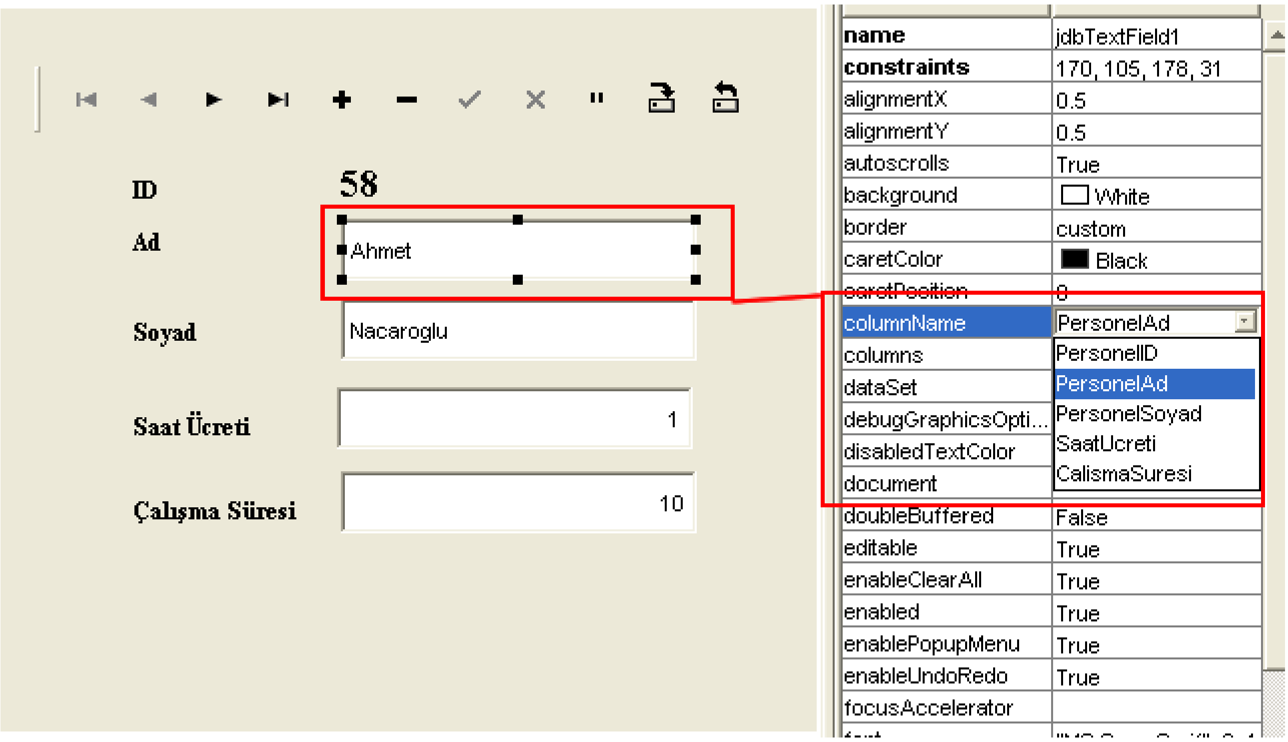


Test Query butonua tıklayarak sorguyu test ettim. Sorgunun başarılı bir şekilde çalıştırılabildiğini gösteren Success ibaresinden sonra işlemlerime devam etmeye hazırdım. Buradaki query seçenekleri arasında önemli olanlarından bir tanesi da, "Execute query immediately when opened" ibaresiydi. Bu seçeneğin işaretli olması ile, uygulama çalışmaya başladığında query' nin otomatik olarak yürütülmesi ve queryDataSet bileşeni ile bağlı olan kontrollerin dolması sağlanıyordu.

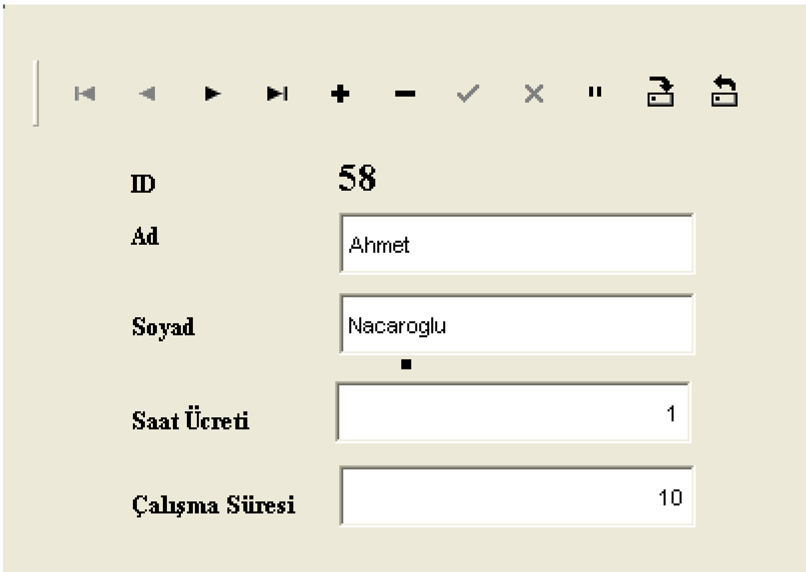
Bu adımlardan sonra, sırada uygulamaya ait Frame bileşenlerinin oluşturulması vardı. Bu amaçla dbSwing sekmesi altındaki bileşenleri kullanabilirdim. İlk olarak, Delphi zamanlarından aklımda kalan bir bileşeni Frame' e yerleştirmeye karar verdim. JdbNavToolBar. Tabi delphi' de bu bileşenin adı farklıydı ama şekil itibariyle neredeyse aynıydı. Bu bileşen, çekilen veri kümesi üzerinde gezinmemi, yeni satırlar eklememi, satırları düzenlememi, güncellememi vb... işlemleri kolayca yapmamı sağlayacaktı.



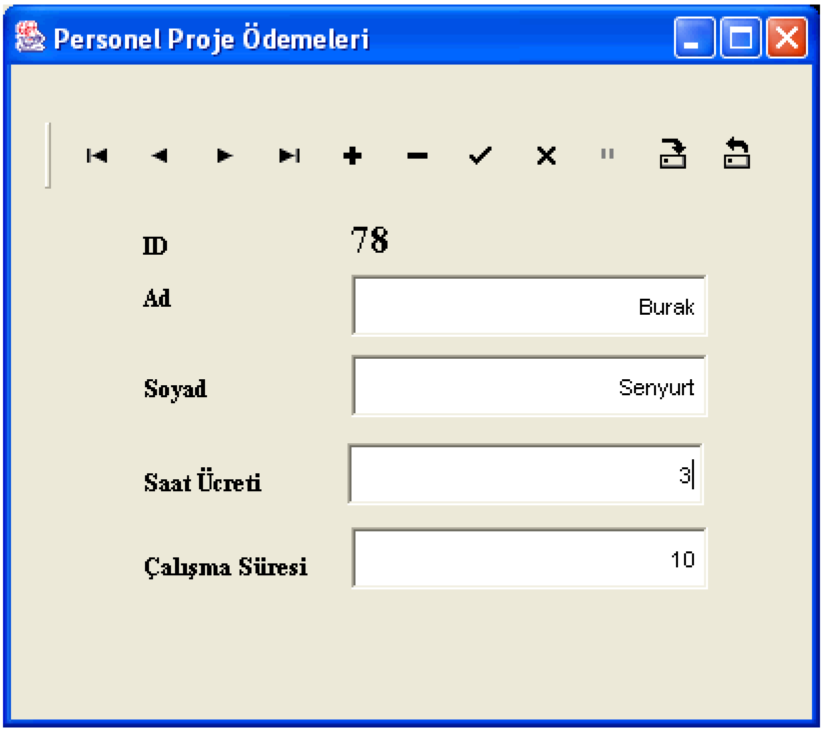
Bileşenimi Frame üzerine ekledikten sonra yapmam gereken tek şey, dataSet özelliğine, queryDataSet1 bileşenini atamak oldu. Böylece, navigator bileşenimi, ilgili veri kümesi ile ilişkilendirmiştim. Bu adımıda tamamladıktan sonra, Frame üzerine Personel tablosundaki satır verilerini gösterecek dbTextField ve dbLabel bileşenlerini ekledim. Bu db bileşenlerininde query özelliklerine, queryDataSet1 bileşenini atadım. Diğer yandan, bu bileşenlerin hangi sütunlardaki verileri göstereceğini belirlemek amacıyla dataColumn özelliklerinede ilgili Column isimlerini atadım. Örneğin, PersonelAd sütunu için;



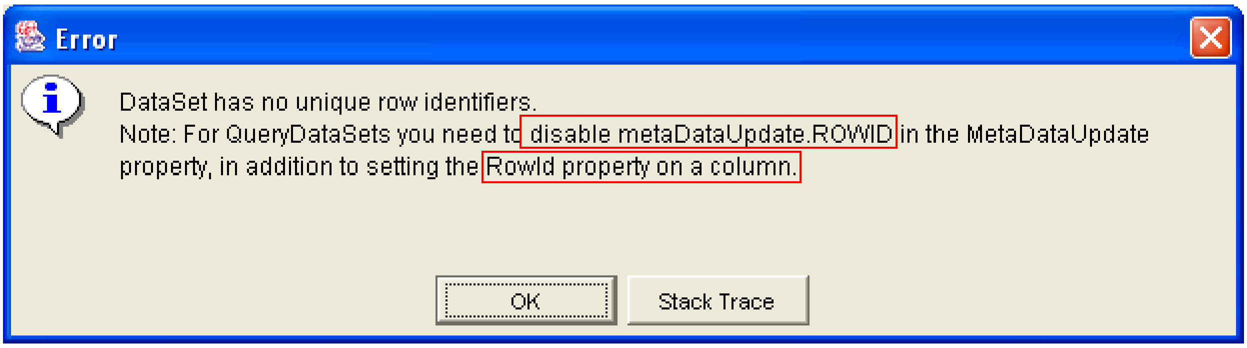
Sonuç olarak Frame tasarımım aşağıdaki gibi olmuştu.



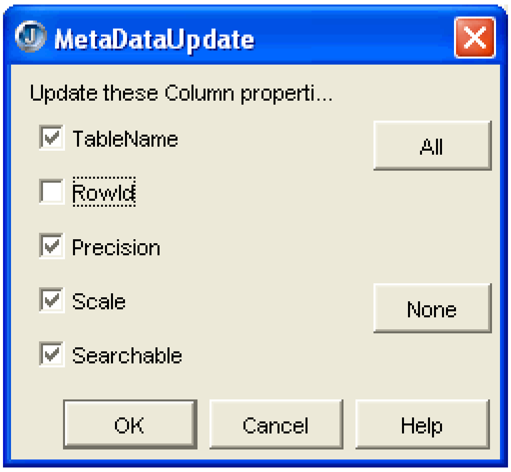
Artık uygulamamı çalıştırabilirdim. Bunun için sabırsızlanıyordum. Hemen işe koyuldum ve uygulamayı Run ettim. Uygulama çalışıyordu, satırlar arasında gezinebiliyordum. Eş zamanlı olarakta, bileşenler içindeki veriler değişiyordu.



Derken hemen yeni bir kayıt eklemek istedim. Bunun için tek yapmam gereken navigator bileşenindeki + işaretine basmaktı. Bunu nerden mi biliyordum. Delphi' deki Navigator bileşeninden tabiki. Ancak, karşıma beklenmedik bir hata mesajı geldi. Sorun Personel tablosundaki PersonelID sütunundaydı. Bu sütunu, Primary Key olarak belirlemiştim ve otomatik artan bir değere göre güncellenmesini ayarlamıştım. Yani, değerini sql sunucusunun kendisi otomatik olarak belirliyor ve benzersiz olmak şartıyla arttırıyordu.

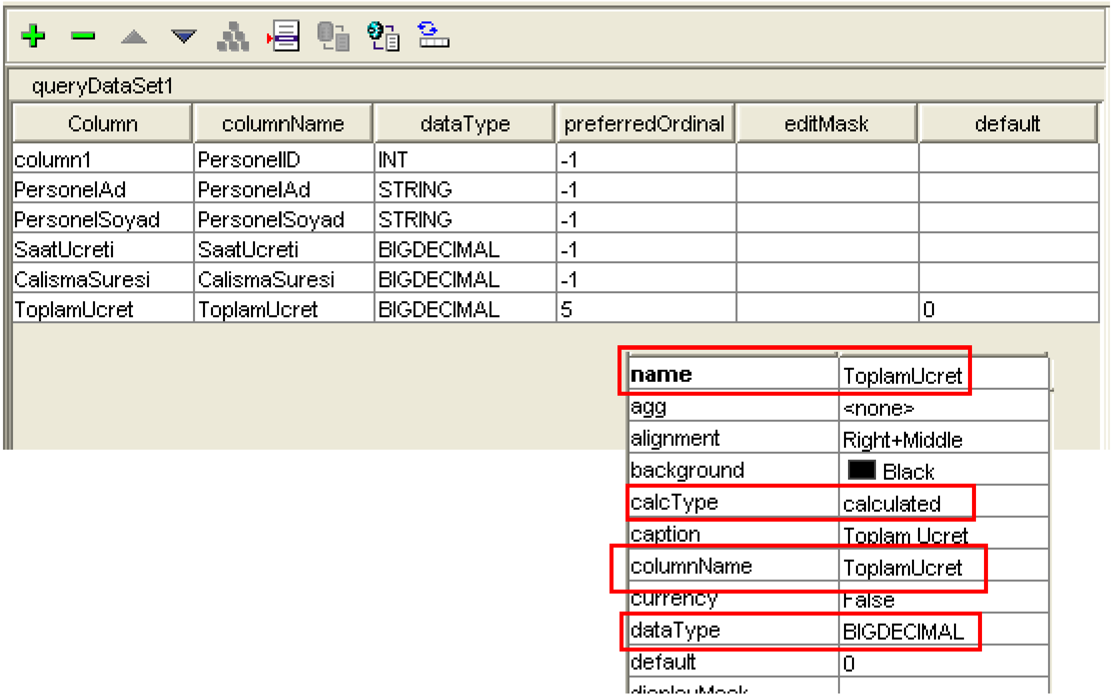


Hemen hata mesajında belirtilen değişiklikleri yaptım. queryDataSet1 bileşeninin, MetaDataUpdate özelliğindeki RowID seçeneğinin işaretini kaldırdım.



Daha sonra ise, PersonelID sütununun özelliklerinden, rowID özelliğinin değerini true olarak değiştirdim. Bu işlemlerden sonra, uygulamamı tekrar çalıştırdığımda artık kayıt ekleyebiliyor, güncelleyebiliyor ve silebiliyordum. Ancak yapmış olduğum bu değişiklikler, sadece queryDataSet bileşenin temsil ettiği veri kümesi üzerinde gerçekleşiyordu. Yani bağlantısız katman üzerinde. Asıl veri tablosuna bu değişikliklerin yansıtılabilmesi için, Navigator bileşenindeki Save Changes ipucu textine sahip butonun tıklanması gerekiyordu.

Şimdi ise sıra calculated field' ın oluşturulmasına gelmişti. Amacım, Saat ücretleri ile çalışma sürelerinin çarpımını veren bir calculated field oluşturmaktı. Bunu gerçekleştirebilmek için öncelikle yeni bir alana ihtiyacım vardı. Bu alan Personel tablosu üzerinde değil, queryDataSet bileşeni üzerinde oluşturulacaktı. Başka bir deyişle çalışma zamanında oluşturulacak ve offline olarak çalışacaktı. Bu alanı ekleyebilmek için, queryDataSet1 bileşenine çift tıklayarak açılan penceredeki + butonunu kullandım.



Yeni alanın columnName özelliği ile ismini, caption özelliği ile başlığını, name özelliği ilede bileşen adını belirttikten sonra, dataType özelliğinide BIGDECIMAL olarak belirledim. Nitekim Sql sunucusunda decimal olarak belirtilen veri tipi, java ortamında BIGDECIMAL olarak eşleştirilmekteydi. Bunlara ek olarak, calcType özelliğinide calculated olarak değiştirmem gerektiğini uygulamayı bu haliyle derlediğim zaman aldığım hata mesajlarından anladım. Bu özelliğide düzenledikten sonra artık tek yapmam gereken, queryDataSet1 bileşeninin calcFields olayını kodlamak olacaktı. Bu olay metodu, calculated field' lar için gerekli hesaplamaların yapıldığı yerde devreye girmekteydi. Buradaki kodları aşağıdaki gibi oluşturdum.

```
void queryDataSet1_calcFields(ReadRow changedRow,
DataRow calcRow, boolean isPosted)
{
  java.math.BigDecimal
saat=chancedRow.getBigDecimal("SaatUcreti");
  java.math.BigDecimal
sure=chancedRow.getBigDecimal("CalismaSuresi");

  java.math.BigDecimal toplamUcret = saat.multiply(sure);

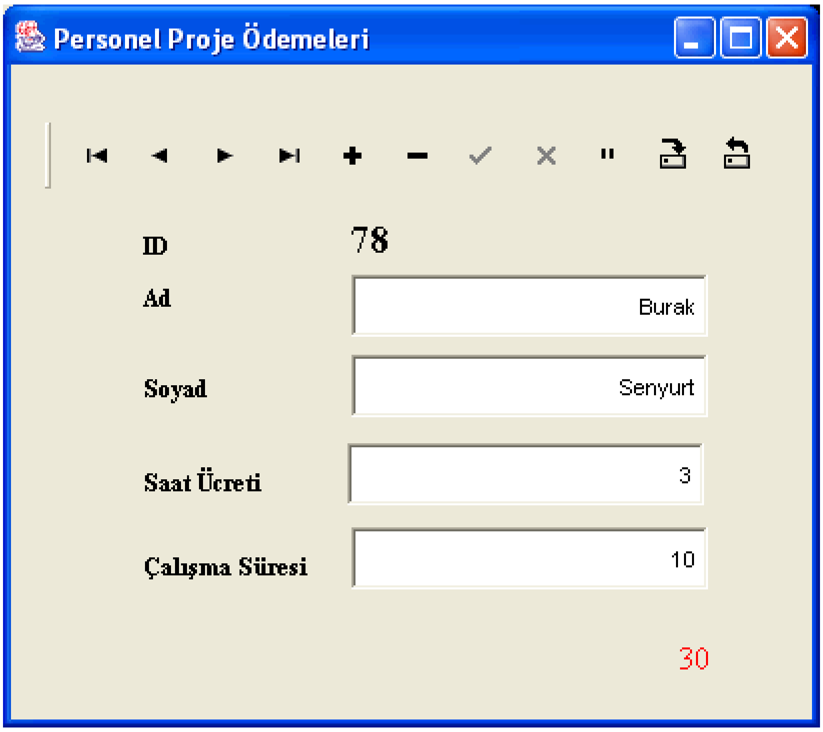
  calcRow.setBigDecimal("ToplamUcret",toplamUcret);
```

```
}
```

Bu kodlarda ilk dikkati çeken nokta, chancedRow parametresi ile güncel satır bilgisine erişilmesiydi. İlk olarak, SaatUcreti ve CalismaSuresi alanlarının değerlerini, ReadRow sınıfından bir örnek olan chancedRow parametresinin getBigDecimal metodu ile, BigDecimal türünden bir değişkene aktarıyorduk. Sonra ise ilginç bir kod satırı çalıştırılıyordu. Nitekim normal şartlar altında iki değişkeni çarpmak istediğimde bildiğim çarpma operatörünü kullanırdım. Ancak burada aşağıdaki kod satırını yazdığımda,

```
java.math.BibDecimal toplamUcret = saat * sure;
```

* operatörünün, BigDecimal tipine uygulanamıyacağı söyleyen bir derleme zamanı hatası alıyordum. Bunun üzerine kaynaklarımı kısa bir süre araştırdığımda, BigDecimal tipindeki değişkenlerden birisine multiply metodunu kullanarak, çarpma işlemini gerçekleştirebileceğimi keşfetmiştim. Calculated field alanının değerine hesaplanan değeri aktarmak için, bu alanı temsil edecek DataRow sınıfının bir örneği olan calcRow parametresinin setBigDecimal metodunu kullandım. Bu işlemlerin ardından, hesaplanan alana ait bilgilerin ekranda gösterilmesini sağlamak amacıyla Frame üzerine, bir JdbLabel bileşeni ekledim. Uygulamamı çalıştırdığımda, ToplamUcret alanının her bir satır için hesaplandığını ve yapılan güncellemelerdende etkilendiğini farkettim. Bu basit uygulama sorunsuz çalışıyordu. Üstelikte 4 satır kod yazmama rağmen.



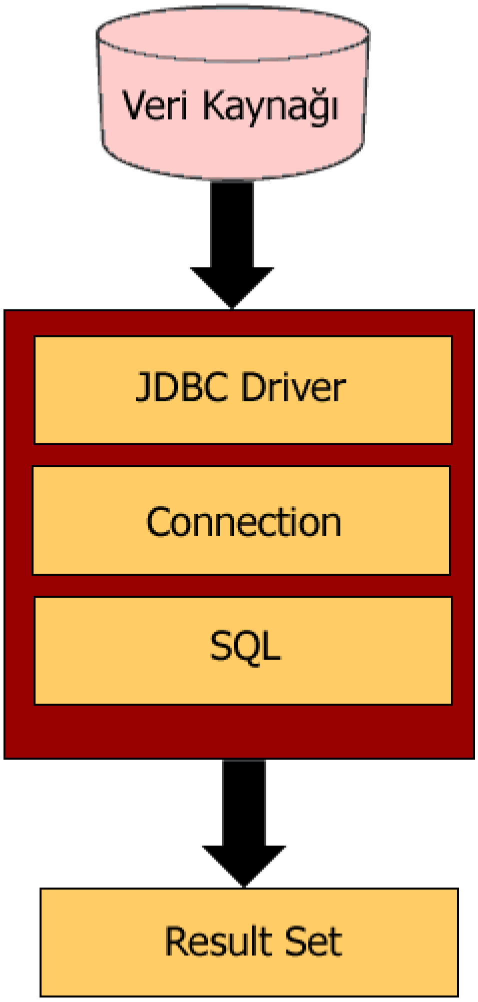
Elbetteki bir veritabanı uygulamasında yapılabilecekler sadece bunlar ile sınırlı değildi. Örneğin, buradaki bileşenleri kullanmadan, JDBC sınıfları yardımıyla bu işlemler nasıl gerçekleştirilebilirdi? Yada bir stored procedure nasıl çalıştırılabilirdi ? Hatta arama, filtreleme gibi işlemler için kodları nasıl düzenlemek gerekiyordu? Bu gibi konuları incelemeye fırsat kalmadan saatin epeyce ilerlediğini ve arkadaşımın ofisi kapamak üzere olduğunu farkettim. Uzun süre boyunca beni izlemiş ve bilgisayar başında nasıl kendimden geçtiğimi seyretmişti. Sonuçta konsantremi bozmak istememişti ama saatin gece yarsını geçtiğinide birisinin bana söylemesi gerekiyordu. "Artık bir sonraki kahve molasına kadar dinlenmelisin" dediğinde, içtiğim kahve bardaklarının masa başında uzunca bir kuyruk oluşturduğunu farketmiştim. İlerleyen haftalar, JDBC ile ilgili yeni araştırmaların habercisiydi.

Bölüm 20: JBuilder ile Database Uygulamaları – 2

Geçtiğimiz hafta boyunca, JBuilder ortamında veritabanı uygulamalarının bileşenler yardımıyla kolay bir şekilde nasıl oluşturulabileceğini inceledim. Bu hafta ise, temel tablo

işlemlerini kod yazmak suretiyle nasıl gerçekleştirebileceğimi araştırdım. JDBC oldukça geniş kapsama sahip bir konuydu. Hatta geniş demek yetmez tam anlamıyla bir okyanus olduğunu söyliyebilirim.

Nasıl ki Ado.Net hakkında yazılmış kalın, devasa kitaplar var ise, JDBC içinde aynı durumun söz konusu olduğunu biliyordum. Ancak en azından elimdeki kaynaklardan faydalanarak temel bir kaç tablo işleminin nasıl gerçekleştirilebileceğinide incelem taraftarıydım. Bana gerekenler temel sql bilgisi ve JDBC' nin bu işler için hangi sınıfları kullandığıydı. Sonuç olarak aşağıdaki tarzda bir vizyon edinmeyi başarabildim. Basit bir veritabanı uygulaması için gerekli unsurlar bunlardı. Bir JDBC sürücüsü, geçerli bir bağlantı, çalışan bir sql cümleciği ve veri kümesini çekebileceğim bir bileşen.



Yapmam gereken son derece basitti. Uygulamam için geçerli bir JDBC Driver' ı yükleyecek sonra bu driver'ı kullanarak,

sistemdeki odbc kaynaklarına bağlanabilecek bir Connection nesnesi oluşturacaktım. Daha sonra bu Connection nesnesini kullanan bir Statement nesnesi yaratacak ve bu nesneyi, tablodaki verileri çekmeme yardımcı olacak bir sql cümleciği ile çalıştıracaktım. Statement nesnesi, Ado.Net' teki SqlCommand nesnesine çok benziyordu. Her ikiside sql cümleciklerini çalıştırmak amacıyla kullanılabiliyorlardı. Sql cümleciğinin çalıştırılması sonucu oluşacak veri kümesini ise ResultSet sınıfından bir nesne örneğine aktaracaktım.

Bu düşünceler eşliğinde kahvemden bir yudum aldım ve uzun süre monitore bakakaldım. Nitekim ne yapacağımı biraz olsun anlamama rağmen nasıl yapacağımı henüz kestirememiştim. Elbetteki, kaynaklarım baş ucumda duruyordu. Ancak özellikle JDBC ile ilgili olan kitaplar tam anlamıyla kalın sayfa sayıları ile korkulu bir rüya gibiydiler. Kaynaklar arasında geziniyor ve en basit haliyle nasıl geliştireceğimi araştırıyordum. O kitap bu kitap derken uzun saatler sonrası bir şeyler çıkartmayı başarmıştım.

Bu araştırma yaklaşık olarak 3 gün, geceli gündüzlü sürdürdüm. Geceleri biraz uyuduğumu itiraf edeyim ama. Halen daha da tam olarak olaya vakıf olabilmiş değildim. Ancak uygulamamı nasıl geliştirebileceğimi en azından biliyordum. Hemen arkadaşımdan ödünç aldığım JBuilder Demo cd' si ile kurduğum JBuilder uygulamasını açtım. Kodlarımı JBuilder üzerinde yazacaktım. Nitekim hem tasarım ile az uğraşmak hemde JBuilder' ın intellisense özelliklerini kullanmak istiyordum. Tabi JBuilder' ın geniş yardım içeriğide işin diğer cazip tarafıydı. Öncelikle yeni bir Proje açtım, bu projeye bir Application ekledim ve temel olarak aşağıdaki gibi bir Frame oluşturdum.



Uygulamada yapmak istediklerim ilk başta, Sql sunucusunda yer alan Personel tablosundaki verileri ResultSet nesnesi içine doldurmak ve sonra bu kayıt kümesindeki satırlar arasında navigasyon tuşları yardımıyla gezinmekti. İlk olarak Doldur başlıklı button bileşenine basıldığında çalışıtırılacak olay kodlarını yazdım.

```
public java.sql.ResultSet rsPersonel;
public java.sql.Connection conPersonel;

void btnDoldur_actionPerformed(ActionEvent e)
{
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        conPersonel = java.sql.DriverManager.getConnection(
"jdbc:odbc:sqlBaglanti");
        java.sql.Statement sqlPersonel =
conPersonel.createStatement(java.sql.ResultSet.TYPE_SCROLL_SENSITIVI
java.sql.ResultSet.CONCUR_UPDATABLE);
        rsPersonel = sqlPersonel.executeQuery("SELECT * FROM Personel");
    }
    catch(Exception ex)
    {
        System.out.println(ex.getMessage());
    }
```

```
}
```

Burada yapılan işlemler şöyleydi. İlk olarak Class sınıfının forName metodu ile, çalışacak uygulama thread' i için bir JDBC Driver' ı yükleniyordu. Ben Sql sunucuma Odbc kaynakları üzerinden bağlanmak istediğim için, JDBC' nin ODBC ile anlaşmasını sağlayacak bir driver' a ihtiyacım vardı. İşte bu amaçla JdbcOdbcDriver' ını kullandım. Artık, uygulama çalıştığında ve Doldur başlıklı butona basıldığında, ilk olarak bu uygulama için sisteme bir JdbcOdbcDriver' ı yüklenecekti. Böylece, çalışan uygulama içinden bu Driver' ı kullanarak ODBC kaynaklarına erişebilecektim. Bu erişimi sağlamak için ise tabiki geçerli bir bağlantıya sahip olmam gerekiyordu. Bunu ise, java.sql paketinde yer alan Connection sınıfı sayesinde gerçekleştirebilirdim.

```
java.sql.Connection conPersonel =
java.sql.DriverManager.getConnection(
"jdbc:odbc:sqlBaglanti");
```

Bu satır sayesinde, sisteme yüklediğim JDBC driver' ını kullanan bir Connection bileşeni elde etmiş oluyordum. Bu Connection bileşeni, ODBC kaynaklarında tanımladığım sqlBaglanti isimli System DSN' ini kullanacaktı. Sonraki satır ise oldukça ilginçti.

```
java.sql.Statement sqlPersonel =
conPersonel.createStatement(java.sql.ResultSet.TYPE_SCROLL_SENSITIVE
java.sql.ResultSet.CONCUR_UPDATABLE);
```

Nitekim burada, Sql cümleciğini çalıştırabileceğim Ado.Net' teki SqlCommand tarzı bir bileşen tanımlanıyordu. Bunun için java.sql paketinden, Statement sınıfı kullanılıyordu. Statement bileşenini oluştururken iki parametre verdim. Bunlar ResultSet sınıfına ait değerlerdi. Amacım uygulama içinde, elde ettiğim kayıt kümesi üzerinde ileri geri hareket edebilmek olduğu için, TYPE_SCROLL_SENSITIVE değerini kullandım. Ayrıca bu değer, yapılan güncellemelerin veya değişikliklerin ResultSet üzerinde hareket ettiğimizde görünmesinide sağlamaktaydı. CONCUR_UPDATABLE değeri ise, update işlemlerinin gerçekleştirilebileceğini belirtmekteydi. Bu değeri tam olarak

anlamış değilim aslında. Sanıyorum ilerleyen zamanlarda kendisini gösterecektir.

Aslında Statement bileşenini parametresizde oluşturabilirdim. Bu durumda, kayıt kümesi varsayılan olarak ileri yönlü okumaya izin veren bir yapıda olacaktı. Dolayısıyla, geriye doğru yapılacak navigasyon hareketlerine izin vermeyecekti. Bu varsayılan değerde, TYPE_FORWARD_ONLY olarak belirtilmekteydi. ResultSet' in davranışı belirleyecek başka alan değerleride vardı. Şu an için bana gerekli olan navigasyona ve insert, update, delete gibi temel tablo işlemlerine izin veren bir ResultSet idi.

ResultSet' in elde edilmesi ise, tanımlanan Statement nesnesinin, executeQuery metodu sayesinde gerçekleştirilmekteydi. Bu metoda parametre olarak çalışmasını istediğim Sql cümleciğini vermiştim. Artık elimde Sql sunucusundaki Personel tablosunda yer alan verlileri uygulama ortamına taşıyan bir veri kümesi vardı. Şimdi uygulamayı test etme zamanı gelmişti. Hemen derleme işlemini gerçekleştirdikten sonra uygulamayı çalıştırdım ve Baglan başlıklı butona bastım. Herhangibir exception çıkmadığından, veri kümesini elde ettiğimi düşünüyordum. Şimdi ise asıl önemli olan ilk satırın verilerini, Frame' deki swing bileşenlerine nasıl dolduracağımdı. ResultSet nesnem üzerinde ileri geri hareket edebilmekteydim. İlk satıra gitmek için first, son satıra gitmek için last, önceki satır için previous ve sonraki satır için ise next metodları vardı. Önce, veri kümesindeki alan değerlerini bileşenlere yazacak ortak bir metod yazdım.

```
void Goster()
{
   try
   {
      this.lbID.setText(rsPersonel.getString("PersonelID"));
      this.tfAd.setText(rsPersonel.getString("PersonelAd"));

this.tfSoyad.setText(rsPersonel.getString("PersonelSoyad"));
      this.tfUcret.setText(rsPersonel.getString("SaatUcreti"));

this.tfSure.setText(rsPersonel.getString("CalismaSuresi"));
```

```
    }
    catch(Exception ex)
    {

    }
}
```

Buradaki kod satırlarında, ilgili bileşenlere setText metodu ile alan değerlerini yüklüyordum. Bunun içinde, ResultSet sınıfının getString metodu ile alan adını kullandım. Tabi almak istediğim değer integer olsaydı getInt metodunu veya tarih olsaydı getDate metodunu vb.' nı kullanabilirdim. Bu işlemlerin ardından, Doldur başlıklı butona ait kod satırlarını ve diğer navigasyon butonlarına ait olay kodlarını aşağıdaki gibi düzenledim.

```
public java.sql.ResultSet rsPersonel;
public java.sql.Connection conPersonel;

void btnDoldur_actionPerformed(ActionEvent e)
{
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        java.sql.Connection conPersonel = java.sql.DriverManager.getConne
"jdbc:odbc:sqlBaglanti");
        java.sql.Statement sqlPersonel =
conPersonel.createStatement(java.sql.ResultSet.TYPE_SCROLL_SENSITIVI
        rsPersonel = sqlPersonel.executeQuery("SELECT * FROM Personel");
        rsPersonel.first();
        Goster();
    }
    catch(Exception ex)
    {
        System.out.println(ex.getMessage());
    }
}

void btnIlk_actionPerformed(ActionEvent e) {
    try
```

```
    {
        rsPersonel.first();
        Goster();
    }
    catch(Exception ex)
    {
    }
}

void btnOnceki_actionPerformed(ActionEvent e) {
    try
    {
        rsPersonel.previous();
        Goster();
    }
    catch(Exception ex)
    {
    }
}

void btnSonraki_actionPerformed(ActionEvent e) {
    try
    {
        rsPersonel.next();
        Goster();
    }
    catch(Exception ex)
    {
    }
}

void btnSon_actionPerformed(ActionEvent e) {
    try
    {
        rsPersonel.last();
        Goster();
    }
    catch(Exception ex)
    {
    }
```

```
}

void this_windowClosing(WindowEvent e) {
    if(rsPersonel!=null)
    {
        try
        {
            rsPersonel.close();
            conPersonel.close();
        }
        catch(Exception hata)
        {
        }
    }
}
```

Uygulamamı çalıştırdığımda herşey istediğim gibiydi. Satırlar arasında gezebiliyordum.



Şimd ise istediğim, verileri güncelleyebilmek ve yeni satır veriler girebilmekti. Kaynaklarımı araştırdığımda, ResultSet sınıfının bu işlemler için kullandığı metodlar olduğunu farkettim. Bu konuları kısa bir sür inceledikten sonra Frame'e güncelleme ve ekleme işlemleri için iki button kontrolü ekledim ve aşağıdaki olay kodlarını oluşturdum.

```
void btngGuncelle_actionPerformed(ActionEvent e) {
   try
   {

rsPersonel.updateString("PersonelSoyad",tfSoyad.getText().toString());
      rsPersonel.updateString("PersonelAd",tfAd.getText().toString());
      java.math.BigDecimal ucret=new
java.math.BigDecimal(tfUcret.getText().toString());
      rsPersonel.updateBigDecimal("SaatUcreti",ucret);
      java.math.BigDecimal sure=new
java.math.BigDecimal(tfSure.getText().toString());
      rsPersonel.updateBigDecimal("CalismaSuresi",sure);
      rsPersonel.updateRow();
   }
   catch(Exception ex)
   {
      System.out.println(ex.getMessage());
   }
}

void btnEkle_actionPerformed(ActionEvent e) {
   try
   {
      rsPersonel.moveToInsertRow();

rsPersonel.updateString("PersonelSoyad",tfSoyad.getText().toString());
      rsPersonel.updateString("PersonelAd",tfAd.getText().toString());
      java.math.BigDecimal ucret=new
java.math.BigDecimal(tfUcret.getText().toString());
      rsPersonel.updateBigDecimal("SaatUcreti",ucret);
      java.math.BigDecimal sure=new
java.math.BigDecimal(tfSure.getText().toString());
      rsPersonel.updateBigDecimal("CalismaSuresi",sure);
      rsPersonel.insertRow();
   }
   catch(Exception ex)
   {
      System.out.println(ex.getMessage().toString());
   }
}
```

ResultSet bileşenin işaret ettiği veri kümesine yeni bir satır eklemek için, önce moveToInsertRow metodu ile yeni bir satır açılıyor ve cursor bu satıra konumlandırılıyordu. Daha sonra ise, updateString ve updateBigDecimal metodları ile, ilk parametre olarak verilen alanlara, ikinci parametredeki değerler yani textField bileşenlerinin içerikleri atanıyordu. Son olarak ise, insertRow metodu çağırılarak oluşturulan yeni satır ResultSet' e ekleniyordu. Güncelleme işlemindede teknik aynıydı. Sadece insertRow yerine updateRow metodu kullanılmaktaydı. Elbette buradaki güncelleme işlemleri için, yine Statement nesnesi kullanılabilirdi. Şöyleki; bu nesne ile sql cümlecikleri çalıştırılabildiğine göre insert ve update sql cümleciklerini çalışıtırarakta ekleme ve güncelleme işlemlerini yaptırabilirdim.

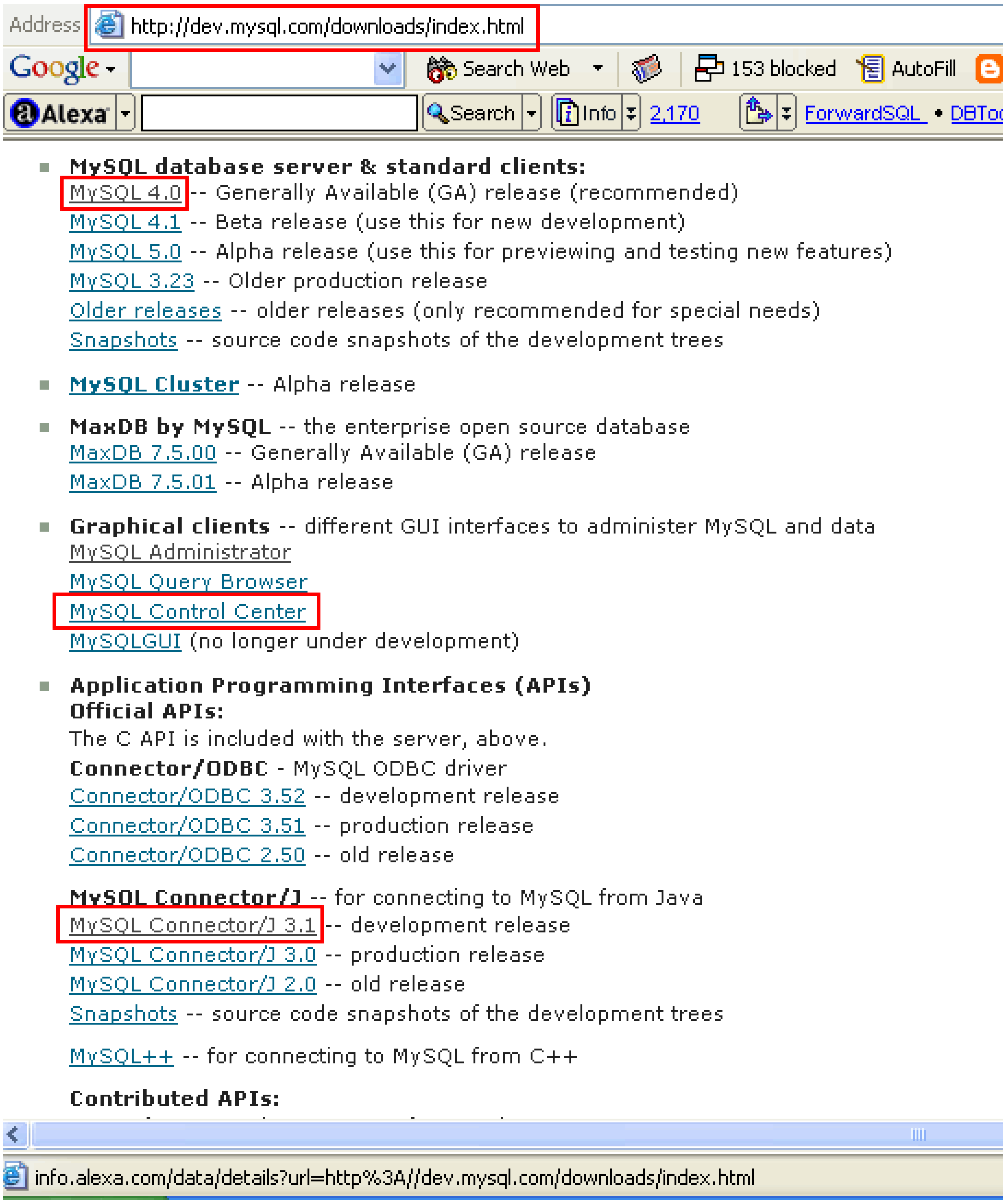
JDBC ile ilgili konular gerçekten saymak ile bitmiyor ve bitecek gibide değil. Bu konu ile ilgili olarak yeni bir 24 kahve molası yapsam az geleceğine inanıyorum. Ancak araştırmaya devam etmekte elbetteki büyük fayda var. Yorucu bir hafta sonrası JDBC ile çok fazla şey yapamasamda en azından biraz kulak dolgunluğu yakalamış durumdayım. Ancak tabiki yeterli değil. Bakalım gelecek kahve molalarında Java okyanusunun hangi derin sularında, ne yükseklikteki çalkantılı dalgalarla boğuşuyor olacağım.

Bölüm 21: Java, MySql Bağlantısı

Java ile uğraşılırda, MySql işin içine katılmaz mı? Yeni nesil uygulamalarda uzun zamandır MySql kullanılmakta olan bir veritabanı sistemi. Çoğunlukla hızlılığı ve performansı ile dikkat çekiyor. Açıkçası bugüne kadar bu veritabanını hiç kullanmamıştım. Çoğunlukla Microsoft Sql Server' ı veritabanı sistemi olarak kullanmaktayım. Ancak internette olsun, yazılı kaynaklarda olsun, Java programlama dilinin yanında mutlaka MySql' e yer verilemekte. Bu belkide bir anlamda, pazar stratejilerinin bir sonucu olsa gerek. Nitekim mysql veritabanı sistemini java platformunda kullanabilmek için gerekli tüm altyapılar kolayca sağlanmış. Neyse, politik davaları ve pazarlama stratejilerini bir kenara bırakıp işimin başına dönsem

daha iyi olacak sanırım. Nede olsa bugüne bugün uygulamacıyım.

Bu hafta amacım, Java uygulamalarında, MySql tablolarını kullanabilmekti. Tabi öncelikle bana neler gerektiğini tespit etmem gerekiyordu. Elbette başrol oyuncusu olan MySql veritabanı sisteminin kurulması ilk sırada yer almaktaydı. Diğer taraftan, Java uygulamalarımda MySql tablolarına erişebilmemi sağlayacak bir veritabanı sürücü api' sinede ihtiyacım vardı. Nitekim, java mysql' e nasıl bağlanacağını bilemezdi. Onun, veritabanlarına bağlanmak için gösterdiği arayüzleri ve bu arayüzlerdeki gerekli metodları yeniden yazacak bir pakete ihtiyacı vardı. Bu düşünceler eşliğinde hemen internette kısa bir araştırma yaptım ve http://dev.mysql.com/downloads/index.html adresine girdim. Bu adreste MySql ile ilgili aradığım herşey mevcuttu.

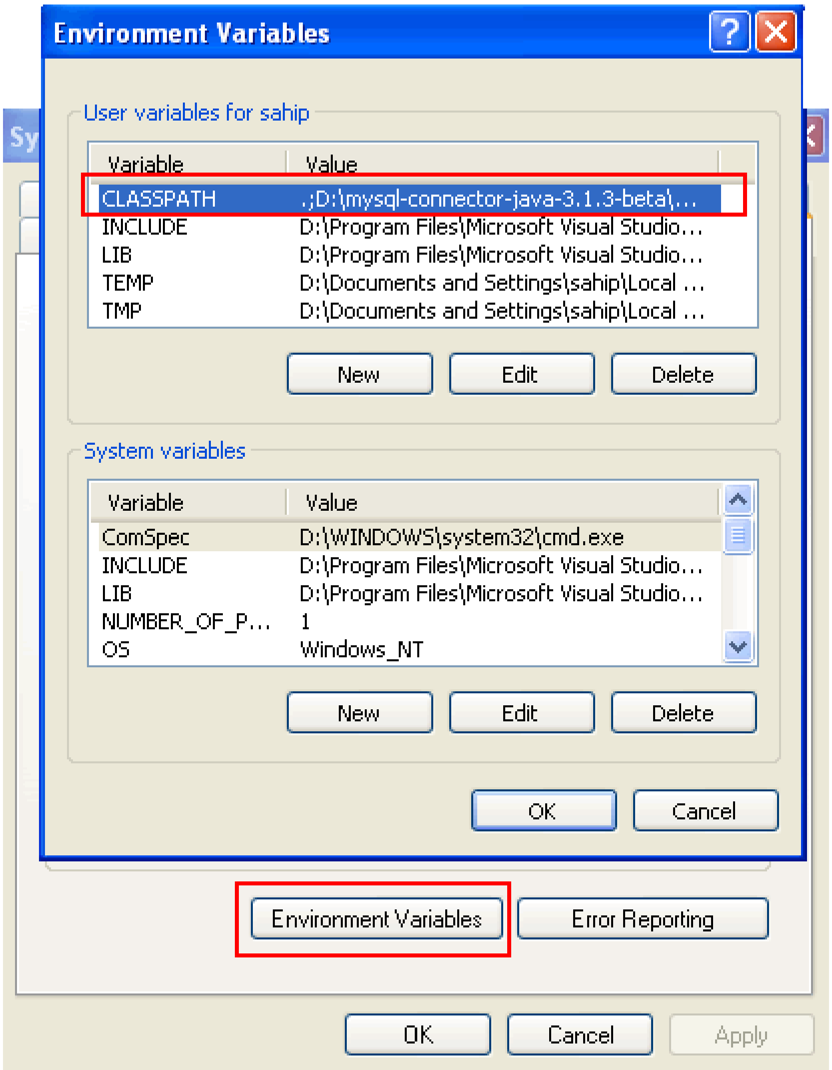


MySql için 4.0 verisyonunu indirmeye başladım. Java platformumda geliştireceğim uygulamaların MySql tablolarını kullanabilmesini sağlayacak sürücü api' si ise, MySql Connector J 3.1 idi. Diğer taraftan, amacım MySql üzerinde sql cümlecikleri ile uğraşmak olmadığı için, Microsoft Sql Server' daki Enterprise Manager' a benzer bir arabirimede ihtiyacım vardı. Araştırmamın cevabı, MySql Control Center uygulamasıydı. Elbette MySql' i kurduktan sonra, sql ifadeleri ile tablolar oluşturabilir, veritabanlarını yönetebilir ve daha pek çok işlemi gerçekleştirebilirdim. Ancak hedefim, bir uygulamacı olarak şu an için tablo oluşturmak gibi işlemlere minimum eforu

harcamaktı. Asıl amaç, MySql tablolarını Java uygulamalarında kullanabilmekti. MySql 4.0’ ı yaklaşık olarak 22 megabyte’ lık boyutu ile sistemime indirdim ve kolayca kurdum. Makinemi yeniden başlattığımda, MySql için gerekli windows servisinin otomatik olarak başlatıldığını ve WinMySqlAdmin uygulamasının Tray’ daki ikonunun yeşil ışığının yandığını gördüm. Bu sistem bir yerlerden tanıdık geliyordu. Baş harfi Microsoft Sql Server Service Manager.



Sırada ise, Java platformum ile MySql arasındaki iletişimi koordine edecek ve sağlayacak paketi yüklemek vardı. MySql Connector paketini sisteme kurduktan sonra hemen yardım dosyalarını üstün körü karıştırdım. Aslında tam olarak ne aradığımı bende kestiremiyordum. Derken gözüme CLASSPATH ile başlayan bir paragraf takıldı. Bu paragrafta kısaca, java’ nın bu api’ yi kullanabilmesi için bir takım ayarların yapılası gerektiğinden bahsediliyordu. Yapmam gereken, ilgili jar paketinin bulunduğu klasörü CLASSPATH tanımlamasına eklemekti. Ben paketi D:\mysql-connector-java-3.1.3-beta\mysql-connector-java-3.1.3-beta\ adresine yüklediğim için classpath tanımlamasınıda buna göre yapmam gerekiyordu. Yani sadece jar paketinin olduğu klasör bilgisini Classpath tanımlamasına ekleyecektim.



Tabi Windows XP kullandığım için bu ayarı, My Computer' e sağ tıklayıp Properties' den Advanced kısmındaki Environment Variables bölümünden yaptım. Böylece sistemde geliştirdiğim tüm java uygulamaları, MySql sunucusuna bağlanmak için gerekli sürücüyü ve bu sürücünün sağladığı üyeleri kullanabilecekti. Sıra gelmişti o heyecanlı ana. Bakalım, bir java uygulaması içinden, MySql' e bağlanabileceğim driver' ı yükleyebilecek miydim? WinMySqlAdmin çalışıyordu. Motorlar hazırdı. Geri sayıma başlamadan önce bir kaç küçük koduda yazmam gerekiyordu elbette. Yoksa roket kalkamadan patlayabilirdi.

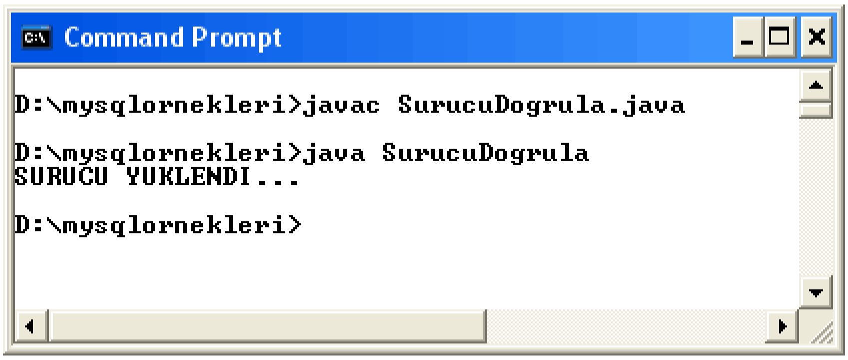
```
import java.sql.DriverManager;

public class SurucuDogrula
{
```

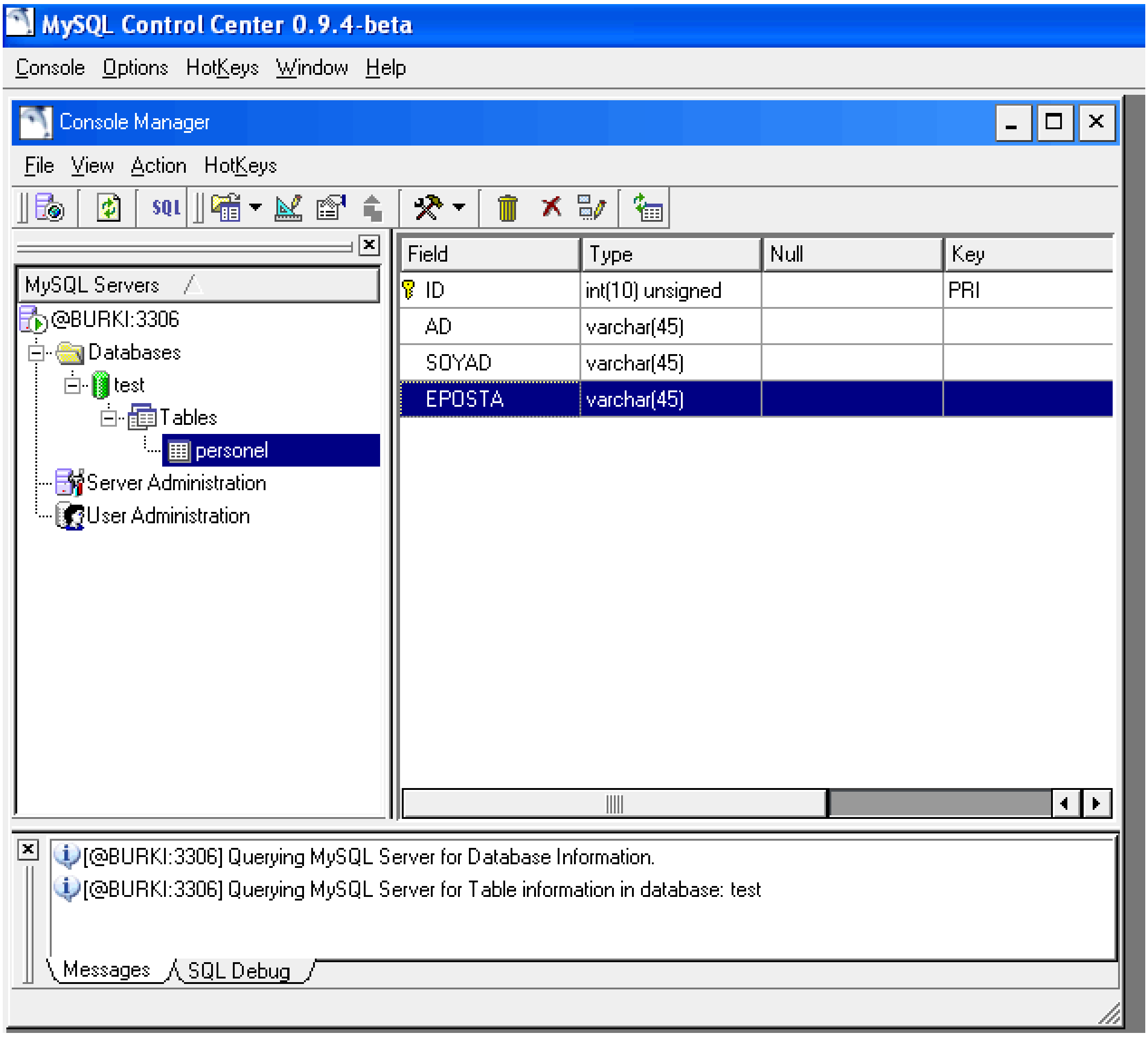
```
   public static void main(String[] args)
   {
      try
      {

Class.forName("com.mysql.jdbc.Driver").newInstance();
         System.out.println("SURUCU YUKLENDI...");
      }
      catch (Exception hata)
      {
         System.out.println("SURUCU
YUKLENEMEDI..."+hata.getMessage());
      }
   }
}
```

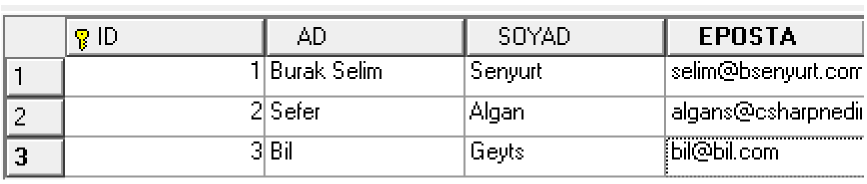
Uygulamayı derleyip çalıştırdım.



Çalışıyordu. Driver başarılı bir şekilde o an çalışan proses için yüklenmişti. Artık daha hızlı adımlarla ilerleyebilirdim. Nede olsa işin en önemli kısmı geride kalmıştı. Artık bana bağlanacağım bir veritabanı ve bu veritabanında yer alacak bir tablo gerekliydi. İşte tam bu sırada aklıma, indirdiğim MySql Control Center programı geldi. Bu programı sisteme kurdum ve ta taaaaa. İşte aradığım arabirim. Artık MySql üzerinde kolayca tablo oluşturabilir ve bu tabloya veriler ekleyebilirdim. MySql kurulduğunda, başlangıç olarak Test isimli bir veritabanınıda sisteme yüklemişti. Şimdi bu veritabanı üzerinde bir tablo oluşturacaktım. Personel isimli bir tablo oluşturdum ve bir kaç alan girdim.



Sonra ise, oluşturduğum tabloya bir kaç satır veri girdim. Herşey Enterprise Manager' dan o kadar tanıdık geliyorduki bu işlemleri yapmam gerçekten çok kısa sürmüştü. Sonuç olarak artık elimde Personel isimli bir MySql tablosu mevcuttu. Üstelik bir kaç satır verisi bilem vardı.



	ID	AD	SOYAD	EPOSTA
1	1	Burak Selim	Senyurt	selim@bsenyurt.com
2	2	Sefer	Algan	algans@csharpnedi
3	3	Bil	Geyts	bil@bil.com

Şimdi Java' yı tekrardan devreye sokmanın sırası gelmişti. İlk yapmak istediğim tabiki MySql sunucusundaki test isimli veritanına bağlanmak ve Personel tablosundaki verileri ekrana

yazdırmaktı. Bunu gerçekleştirebilmek amacıyla uygulama kodlarımı aşağıdaki şekilde geliştirdim.

```
import java.sql.*;

public class SurucuDogrula
{
    public static void main(String[] args)
    {
        try
        {

Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection conTest =
DriverManager.getConnection("jdbc:mysql://localhost/test");
            Statement komut= conTest.createStatement();
            ResultSet rs = komut.executeQuery("SELECT * FROM
Personel Order By ID");
            while(rs.next())
            {
                System.out.println("ID "+rs.getString("ID"));
                System.out.println("ISIM "+rs.getString("AD"));
                System.out.println("SOY ISIM
"+rs.getString("SOYAD"));
                System.out.println("E POSTA
"+rs.getString("EPOSTA"));
                System.out.println("-------------------------");
            }
            conTest.close();
        }
        catch (Exception hata)
        {
            System.out.println("SURUCU
YUKLENEMEDI..."+hata.getMessage());
        }
    }
}
```

Uygulama kodlarında öncekilerden farklı pek bir şey yoktu aslında. Herşeyden önce en önemli fark, veritabanı sürücüsünün MySql için geliştirilmiş olduğu ve bu nedenlede, Connection

nesnesi oluşturululurken buna uygun bağlantı katarının girildiğiydi. Yerel sunucuda bulunan test isimli veritabanına bir bağlantı açmak istediğimden, bağlantı ifadesi jdbc:mysql://localhost/test şeklinde olmalıydı. Sonraki adımlarda, sadece tablodaki verileri çekmek için kullanacağım sql sorgusunu çalıştıracak bir Statement nesnesi oluşturmuş ve sorgu sonuçlarını bir ResultSet kümesine alarak, bu kümedeki satırları while döngüsünde gezmiştim. Alan değerlerini getString("Alan adı") tekniği ile alıyordum. Elbetteki döngünün, veri kümesindeki tüm satırlarda gezmesi için next metodu kullanılıyordu. Herşey bittikten sonrada kaynakları geri iade etmek amacıylada, bağlantı nesnesini kapatıyordum.

Sırada satır ekleme, satır silme ve güncelleme gibi temel işlemler vardı. Bu kez ResultSet' e ait metodlar yerine sql cümleciklerini kullanmaya karar verdim. Statement sınıfı bu işlemler için, executeUpdate isimli bir metod sunuyordu. Bu metod ile, tablo üzerinde yapılacak güncellemeler, satır ekleme işlemleri ve satır silme işlemleri gerçekleştirilebilmekteydi. Tabloya satır girilmesi ile işleme başladım. Bunun için, try bloğu içerisine aşağıdaki kod satırlarını ekledim.

```
System.out.println("-------------------------");
int eklenen=komut.executeUpdate("INSERT INTO Personel
(AD,SOYAD,EPOSTA) VALUES
('Deneme','Deneme','Posta1@posta.com')");
System.out.println(eklenen+" SATIR EKLENDI...");
```

İşin tek esprisi, sql cümleciğinin, Statement sınıfının executeUpdate metodu ile çalıştırılmasıydı. Bu metodun geri dönüş değeri ise int tipindendi ve işlem sonucu etkilenen satır sayısını vermekteydi. Uygulamayı tekrardan derleyip çalıştırdığımda başarılı bir şekilde satırın Personel tablosuna eklenmiş olduğunu gördüm.

	ID	AD	SOYAD	EPOSTA
1	1	Burak Selim	Senyurt	selim@bsenyurt.com
2	2	Sefer	Algan	algans@csharpnedir.com
3	3	Bil	Geyts	bil@bil.com
4	4	Deneme	Deneme	Posta1@posta.com

Sırada güncelleme işlemi vardı. Tek yapmam gereken sql cümleciğini uygun şekilde değiştirmekti. Burada, Personel tablosundaki tüm satırların EPOSTA alanlarının değerlerini değişitiriyordum. Komutun çalışması sonucu bu işlemden etkilenen satır sayısı ise, executeUpdate metodunun sonucu olarak ortama aktarılmaktaydı.

```
System.out.println("-------------------------");
int eklenen=komut.executeUpdate("UPDATE Personel SET
EPOSTA='admin@admin.com'");
System.out.println(eklenen+" SATIR GUNCELLENDI...");
```

Silme işlemi de extra bir efor gerektirmemekteydi. Sql diline biraz olsun aşina olduğum için kodda ne tür değişiklik yapmam gerektiğini biliyordum. Burada örnek olarak ID alanı 4 olan satırı tablodan çıkartıyordum.

```
System.out.println("-------------------------");
int eklenen=komut.executeUpdate("DELETE From Personel
WHERE ID=4");
System.out.println(eklenen+" SATIR SILINDI...");
```

Tüm bu işlemler elbetteki MySql sunucusunda yer alan, Personel tablosunu doğrudan etkilemekteydi. Artık MySql sunucusuna ait tablolarıda kullanabilmekteydim. Veritabanları ile ilgili konular tabikide çok geniş ve kapsamlı. Bu konu ile ilgili olarak yazılmış binlerce sayfalık kitaplar olduğunu, bu kitapları gördüğümü ve onlara korku dolu bakışlarla baktığımı söylemek isterim. Ancak temel olarak yapabileceklerimi bilmek bile son derece güzel. Hiç olmassa şimdilik başımın çaresine bakabilirim. Bakalım kahve kokusunun zihnimde oluşturduğu görüntüler, beni başka hangi konulara sürükleyecek.

Bölüm 22: Java ile Grafik Çizim

Geçtiğimiz hafta boyunca, Java dili ile fazla ilgilenemedim. Nitekim vaktimin büyük çoğunluğunu Whidbey' i incelemekle geçirmiştim. Aslında yazın bu sıcak dönemlerinde, beni şöyle

rahatalacak, fazla terleymeyecek çalışmalar yapmak istiyordum. Whidbey beni bir nebze olsa rahatlatsada, klimanın verdiği ferahlığı sağlayamamıştı. Bana biraz eğlenceli ve eğlenceli olduğu kadarda işe yarayacak bir konu gerekiyordu. Sonunda, Java programlama dili ile, grafiksel çizimlerin nasıl yapıldığını araştırmaya karar verdim. Zor olmayan, sıkıcı olmayan hatta zaman zaman işe yarar bir şekil oluşturabilmek için eski matematik bilgilerimi hatırlamama yol açan bu konu benim için yeteri kadar eğlenceli ve güzeldi.

Elbette, bir programlama dili ne kadar güçlü olursa olsun, sağladığı grafiksel kütüphanalerin kabiliyetleri, sıradan bir tasarım programının yerini tutamazdı. Ancak insan durup düşündüğünde, bu tip grafik programlarının oluşturulmasında java, C# gibi dillerin kullanılabileceğini kolaylıkla anlayabilir. Sonuç olarak, bir grafik programında mouse ile, toolbar' dan seçtiğimiz bir şekli kolayca oluşturabiliriz. Mouse ile sürükleme bir olaydır. Seçilen şekle göre ekranda bir vektör grafiğin oluşmasıda, dilin sağladığı grafik kütüphaneler ile mümkün olabilir. Olayı dahada sofistike düşündüğümde, C# ile veya Java ile yazılmış, haritacılık, şehir planlama gibi programların olduğunu da biliyordum. Hatta böyle bir programı iş başındaykenden inceleme fırsatı bulmuştum.

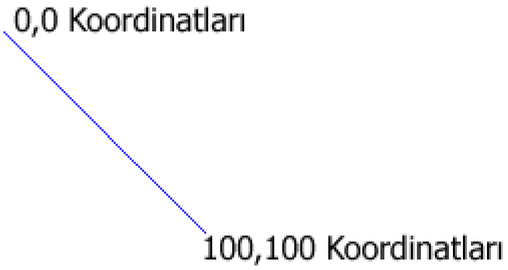
Sonuçta, eğlenceli olan grafik nesneleri aslında büyük çaplı projelerdede temel yapı taşları olarak rol alabilirlerdi. Kendimi bu düşünceler eşliğinde gaza getirdikten sonra, şöyle ağız tadıyla web sayfasına bir çizik atayım dedim. Yapacağım işlemler son derece kolaydı. Bir applet tasarlayacak ve bu sayede bir web sayfasına, çeşitli grafik metodları yardımıyla vektör şekiller çizebilecektim. Bunun için aşağıdaki gibi bir java kaynak kod dosyasını oluşturdum.

```
import java.awt.*;
import java.applet.Applet;

public class Grafikler extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.BLUE);
```

```
        g.drawLine(0,0,100,100);
    }
}
```

Ekrana bir çizgi çizmek için, Applet' in paint metodunu kullandım. Burada, çizgiyi Graphics sınıfının, drawLine metodu ile oluşturdum. Metod 4 parametre alıyordu. İlk ikisi x ve y koordinatlarını, son ikisi ise, çizginin bittiği yerin x ve y koordinatlarını vermekteydi. Ayrıca, çizgiyi mavi renkte boyamak istediğimden, Graphics nesnesine setColor metodu ile Color numaralandırıcısından BLUE değerini atadım. Sonuç aşağıdaki gibiydi.

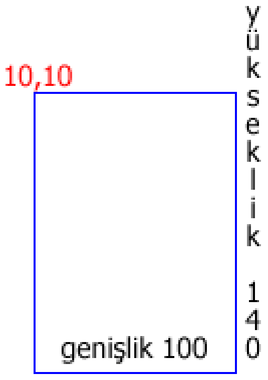


Elbette Graphics sınıfından çizebileceğim şekiller sadece basit bir çizgiden ibaret olamazdı. Daha pek çok şekil vardı. İçi dolu olanlar veya içi boş olanlar gibi. O halde yapmam gereken ortadaydı. Çizebileceğim her şekli deneyecektim ve sonuçlarını görecektim. Bu aynı zamanda benim için bir rehber olacaktı. Çizim kodu ve altında şeklin görüntüsü. Hemen parmakları sıvadım ve klavyemin tuşlarını aşındırmaya başladım. Öncelikle temel şekillerle işe başlama taraftarıydım. Yani dörtgenlerden.

```
import java.awt.*;
import java.applet.Applet;

public class Grafikler extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.BLUE);
        g.setFont(new Font("Verdana",Font.BOLD,12));
        g.drawString("ICI BOS DORTGEN",0,165);
```

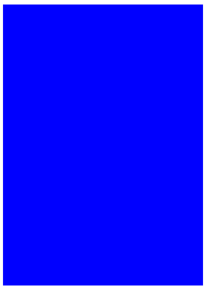
```
        g.drawRect(10,10,100,140);
    }
}
```



ICI BOS DORTGEN

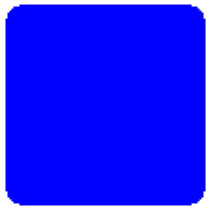
drawRect metodu 4 parametre almaktaydı. İlk ikisi, dörtgenin yerleştirileceği X ve Y koordinatlarını belirtirken, sonraki iki parametrede genişlik ve yüksekliği belirtmekteydi. Bir dörtgenin içinin dolu olmasını veya kenarlarının yuvarlatılmış olmasını isteyebilirdik. Bunun içinde değişik metodlar vardı. Örneğin fill ile başlayan metodlar içi dolu şekiller oluştururken, draw ile başlayanlar içi boş şekiller oluşturmaktaydı. Hemen, diğer olası dörtgen şekillerinide çizmeye çalıştım ve aşağıdaki örnek kodları oluşturdum.

```
g.fillRect(10,10,100,140);
```



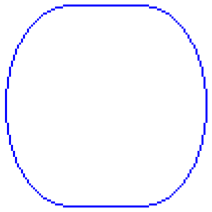
ICI DOLU DORTGEN

```
g.fillRoundRect(10,10,100,100,16,16);
```



ICI DOLU VE KENARLARI YUVARLATILMIS DORTGEN

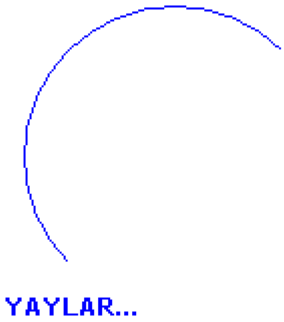
```
g.drawRoundRect(10,10,100,100,64,128);
```



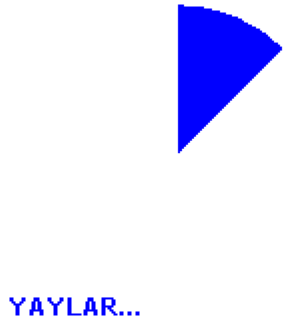
ICI BOS VE KENARLARI YUVARLATILMIS DORTGEN

Round tipteki dörtgenlerin kenarları yuvarlatılırken son iki parametre kullanılmakta. Bunlardan ilki, yayın genişliğinin piksel cinsinden değerini belirtirken ikinciside yayın yüksekliğini belirtiyor. Dörtgenlerden sonra sıra yaylara gelmişti. Lisans eğitimim sırasında, yaylarla epeyce haşırneşir olmuştum. Ancak genelde yayları elle kağıtlara çizmek yerine, yaylar ile ilgili teoremlerin ispatlarının yapılması ile uğraşmıştım. Sırf bu nedenle, ilk ve orta okulda çok iyi olan resim çizme kabiliyetimin yok olduğunu söyleyebilirim. Nitekim, artık şekil çizmeye çalıştığımda o şekle matematiksel bir gözle bakıyor ve olabilecek çeşitli ispatları ve teoremleri hatırlıyorum. Neyse, konuyu ve kafayı fazla bulandırmadan yay çizme işlemine girişsek hiç fena olmaz sanırım.

```
g.drawArc(10,10,150,150,45,180);
```

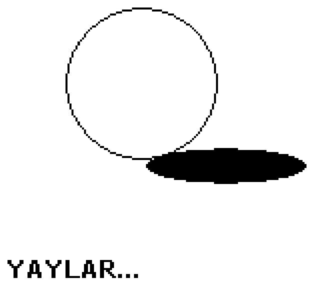



```
g.fillArc(10,10,150,150,45,45);
```

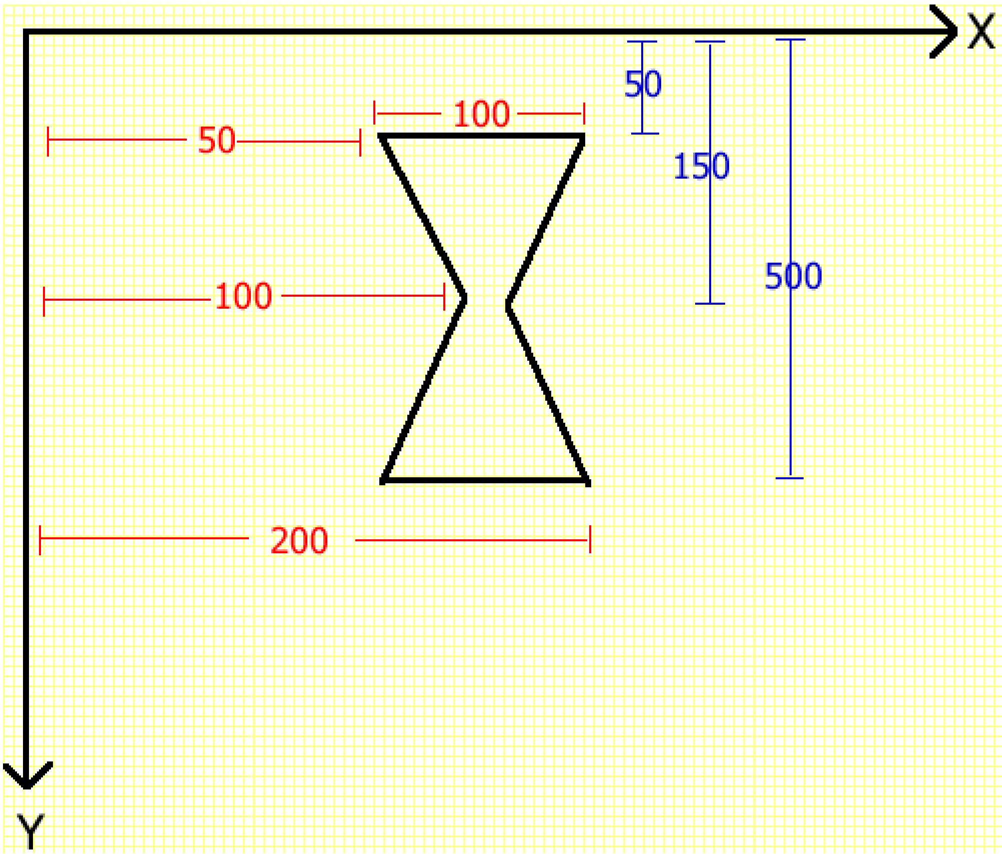


Yayların çizimindeki en önemli nokta son iki parametreydi. Bu parametrelerden ilki, başlangıç açısını derece cinsinden belirtirken, ikinci parametre yayın oluşturacağı açıyı derece cinsinden belirtmekteydi. Tabiki, içi dolu yay aslında bir daire diliminden başka bir şey olmamaktaydı. Ancak daireleri çizmek içinde başka metodlar vardı. Daireler oval şekillerin çizildiği fillOval yada drawOval metodları ile elde edilebilirdi. Daire olması için, genişlik ve yükseklik değerlerinin eşit olması yeterliydi. Bu şekilleride aşağıdaki kod satırları ile test ettim.

```
g.setColor(Color.BLACK);
g.setFont(new Font("Verdana",Font.BOLD,12));
g.drawString("YAYLAR...",0,165);
g.fillOval(70,100,80,18);
g.drawOval(30,30,75,75);
```



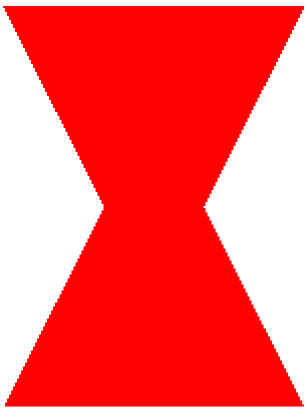
Sırada daha komplike bir şekil olan poligonlar vardı. Bir poligon çizebilmek için, drawPolygon yada fillPolygon metodlarından birisini kullanabilirdim. Aralarındaki tek fark birisinin içinin dolu ötekisinin ise boş oluşuydu. Hangileri tahmin edin bakalım :) Elbette bir poligon oluşturabilmek için bir takım verilere ihtiyacım vardı. Herşeyden önce poligonların köşe sayıları belli değildi. Dolayısıyla bu köşelerin x ve y koordinatlarını belirleyecek iki integer diziye ihtiyacım olacaktı. Bu dizilerden birisi, x koordinatlarını diğeri ise y koordinatlarını taşımalıydı. İşte bu anda, kalem kağıda sarıldım ve acaba bir kum saatinin sembolik resmini çizebilir miyim diye düşünmeye başladım. Oturup ciddi ciddi, kağıt üzerinde, bir kum saatinin iki boyutlu görüntüsüne ait köşe koordinatlarını, ekranın ordinat sistemine göre çıkarmaya çalıştım. İşte kağıttaki çalışmamım güzelim Fireworks grafik programı ile şematize edilişi.



Sonuç olarak aşağıdaki kodlar bana izleyen şekli verdi. Ehhh işte. Biraz da olsa kum saatini andırıyor.

```
int xKoordinatlari[]={50,200,150,200,50,100};
int yKoordinatlari[]={50,50,150,250,250,150};

g.fillPolygon(xKoordinatlari,yKoordinatlari,6);
```



Artık tüm şekilleri öğrendiğime göre, bunları harmanlayıp beni daha çok neşelendirecek bir uygulama yazabilirdim. Örneğin,

hepimizin kağıda çizmekte hiç zorlanmayacağı, hatta patatesler ve bir kaç kibrit çöpü ile birlikte yapabileceği bir çöp adamı, Java' daki grafik metodlarını kullanarak çizmek ne kadar zor olabilirdi ki? Doğruyu söylemek gerekirse o basit çöp adamı oluşturabilmek için bir süre çalışmam gerekti. Kağıt üzerinde matematiksel olarak koordinatların belirlenmesi, hangi şeklin nereye geleceği derken saat sabahın üçü oluvermişti bile. Ancak bir kaç dakikalık bu eziyet sonrasında aşağıdaki program kodları sayesinde oldukça ilginç bir eser ortaya çıkartabilmiştim. Belki Da Vinci kadar iyi değildi. Ancak yinede modern sanatın bir görüntüsüydü.

```
import java.awt.*;
import java.applet.Applet;

public class Grafikler extends Applet
{
   public void paint(Graphics g)
   {
      g.setColor(Color.RED);
      g.setFont(new Font("Verdana",Font.BOLD,12));
      g.drawArc(20,10,20,20,45,75);
      g.drawArc(50,10,20,20,45,75);

      g.setColor(Color.blue);
      g.drawLine(20,1,20,6);
      g.drawLine(25,1,25,7);
      g.drawLine(30,1,30,7);
      g.drawLine(35,1,35,8);
      g.drawLine(40,1,40,8);
      g.drawLine(45,1,45,9);
      g.drawLine(50,1,50,8);
      g.drawLine(55,1,55,8);
      g.drawLine(60,1,60,7);
      g.drawLine(65,1,65,7);
      g.drawLine(70,1,70,6);

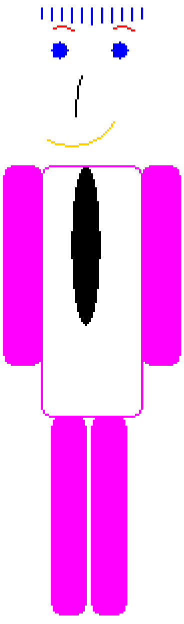
      g.fillOval(25,18,9,9);
      g.fillOval(55,18,9,9);
```

```
        g.setColor(Color.BLACK);
        g.drawArc(37,30,15,45,130,60);

        g.setColor(Color.orange);
        g.drawArc(10,20,50,50,-120,90);

        g.setColor(Color.black);
        g.fillOval(35,80,15,80);

        g.setColor(Color.MAGENTA);
        g.fillRoundRect(1,80,20,100,10,10);
        g.drawRoundRect(20,80,50,125,10,10);
        g.fillRoundRect(70,80,20,100,10,10);
        g.fillRoundRect(25,205,18,100,10,10);
        g.fillRoundRect(45,205,18,100,10,10);
    }
}
```



Şimdi gelde, şu grafik programlarını, harita programlarını nasıl yazıyorlar düşün. Düşünmeden edemedim. Allah bu tip grafik tabanlı yazılımları Java veya C# gibi nesne yönelimli diller ile geliştirilenlere sabır versin. Bende bir kaç have finacanı ile ancak yukarıdaki kadar garip bir adam elde edebildim.

Bölüm 23: Servletler ile Dinamik Etkileşimli Web Sayfaları

Düşünüyorumda, internet yayıldı yayılalı, dünyanın çehresi değişti. Internet' in ilk yıllarında, sadece belirli sayıda sitede, basit html kodları ile oluşturulmuş sayfalar yer alırdı. Örneğin mezun olduğum bölüme ait web sayfalarını o yıllarda bir arkadaşım bitirme tezi olarak sunmuştu. Sayfadaki tek atraksiyon, bölümün öğretim görevlilerinin isimlerine tıklandığında, onlar hakkında bir adet resim ve çeşitli bilgilerin bulunduğu metinsel ağırlıklı sayfalara giden linklerin yer almasıydı. Ha birde mail gönderebileceğimiz linkler vardı. O tarihlerde bu sayfayı arkadaşım notepad üzerinde html takıları ile gerçekleştirmiş ve tez ekibini oldukça etkilemişti.

Ancak kısa sürede internet üzerindeki web içeriği inanılmaz ölçüde çehre değiştirdi. Bugün müşteriye yönelik, şirketlere yönelik ve daha pek çok alanda geliştirişmiş sayısız internet sitesi var. Gelişen teknolojiler ile birlikte bu sitelerin, kullanıcıları ile olan etkileşimleride değişiklik gösterdi. Öyleki, artık dinamik etkileşim söz konusu. Kullanıcının taleplerini işleyen ve cevaplar üreten, bu cevaplara göre sayfaların dinamik olarak oluşturulmasını sağlayan teknikler söz konusu. Aslında kısa bir süre düşününce gelinen noktanın gerçekten göz kamaştırıcı olduğunu söylemek lazım. Nitekim her ne zaman Amazon' dan bir kitap almak istesem, baktığım ürünün yanında mutlaka, "bu ürünü alanlar bunlarıda aldı" gibisinden yönlendirici içerikler görmekteyim. Hatta bazen kişiye özel indirimlerin yapıldığıda oluyor. Bu dinamik etkileşime verilebilecek basit ve önemli örneklerden birisi aslında. Çünkü, web sayfasının görünümü ve içeriği benim vermiş olduğum taleplere göre, dinamik olarak değişmekte. Benim için sayfada görünen tavsiye ürünler, başka bir kullanıcı için başka ürünler olabilir.

Peki acaba, dinamik etkileşimli web sayfaları nasıl geliştirilebilir? İlk olarak, dinamik etkileşimi gerçekleyebilecek en iyi şart güçlü, esnek bir programlama dilinin olmasıdır. Bugün itibariyle, ben bu tip işleri Asp.Net platformunda C# programlama dilini kullanarak gerçekleştirmekteyim. Acaba eskiden nasıldı?

Kaynaklarımı araştırdığımda, ilk olarak perl dilinin cgi ile yan yana anıldığını keşfettim. Daha önce bu konular ile hiç uğraşmadığımdan tam olarak nasıl kullanıldıklarını bilmiyorum. Ancak dinamik etkileşime izin veren bir yapıyı sağladıklarını anlayabiliyordum. Nitekim, cgi sonrasında servlet, jsp, asp gibi modellerin çıkması, cgi' ın bir takım sınırlamalarının olduğunu göstermekte. Bunu kaynaklarımdan kısaca araştırdığımda, cgi ile geliştirilen sunucu taraflı web sitelerinde, çökme probleminin daha sık yaşanmasına neden olabilecek bir tekniğin kullanıldığını öğrendim. Öyleki, cgi ile geliştirilen bir web sitesinde, sunucu taraflı kodlama için genellikle perl dili kullanılarak yazılan programlam mevcut. Sayfaya yapılan her bir kullanıcı talebi için, bu programlar ayrı birer süreç oluşturacak şekilde çalıştırılıyormuş. Buda aynı anda siteye bağlanabilecek kullanıcı sayısında bir sınırlama getiriyormuş. Dolayısıyla, hastalığın tedavisi yine ve her zamanki gibi ilk olarak Sun firmasından servlet' ler ile gelmiş.

Bir servlet aslında, java byte kodlarından oluşan bir sınıf nesnesinden başka bir şey değil. Sunucu taraflı olan bu nesneler, java dili ile yazılabildiğinden, web sayfasına müthiş bir programlama kabiliyeti getirmekte. Servlet' lerin tek dezavantajı okuduğum kadarı ile, html içeriği ile kodların ayrı olarak yazılmasını gerektirmesi. Bu html içeriği ile sunucu taraflı kodların aynı asp sayfalarında olduğu gibi bir arada kullanılabildiği jsp' lerin doğmasına yol açmış. Gerçi şu anda, Asp.Net ile uygulama geliştiriren sürekli olarak code-behind tekniğini kullanmaktayım. Yani web sayfasına ait htm içeriği ile, C# uygulama kodlarım tamamen farklı dosyalarda duruyor. Bu işimi dahada kolaylaştırıyor. Ancak tabiki servlet' ler Asp.Net ortamında olduğu gibi bir takım kolaylıklar ve güçler kazandıramamış zamanında. Tabi bu noktada aklıma gelen soru jsp' lerin servlet' lere karşı bir üstünlükleri olup olmadığı. Gerçek şu ki, ikisi arasındaki tek fark kodlamaların yazılış yerleri.

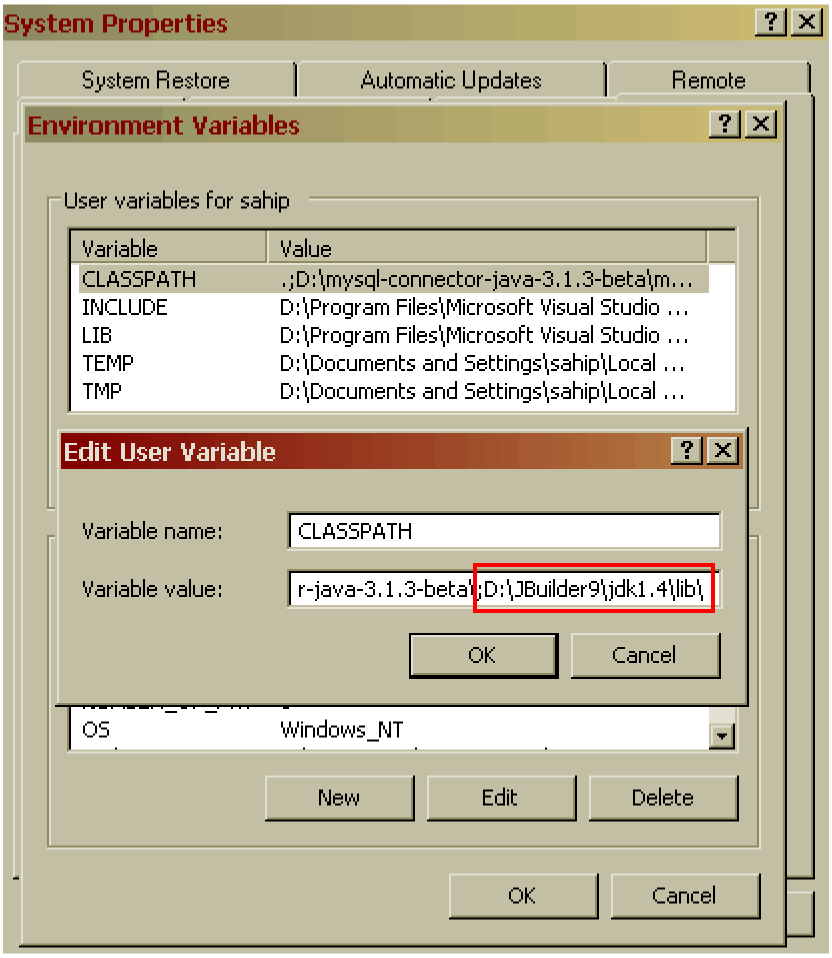
Edindiğim bu kısa bilgilerin ardından hemen servlet' lerin nasıl uygulandığını denemeye karar veriyorum. Lakin, bana gereken bir takım materyaller var. Öncelikle, Asp.Net ile geliştirdiğim web sayfalarından biliyorum ki bana sanal bir web sunucusu lazım. İşte bu noktada devreye adını sıkça duyduğum, Apache

Tomcat sunucuları giriyor. Lakin daha basit ve işin uzmanından bir çözüm var. JSWDK. Jswdk, aslında java servletlerini ve jsp sayfalarını barındırıp çalıştırabilen bir web sunucusu. Geliştiricisi ise tabiki Sun. Bu nedenle ilk olarak sistemime, JSWDK' yı kurmam gerekiyor. Bu amaçla, Sun' ın internet sitesinden, http://java.sun.com/products/servlet/archive.html linkinden (bu gün itibariyle) JSWDK' nın 1.0.1 versiyonunu indirdim.

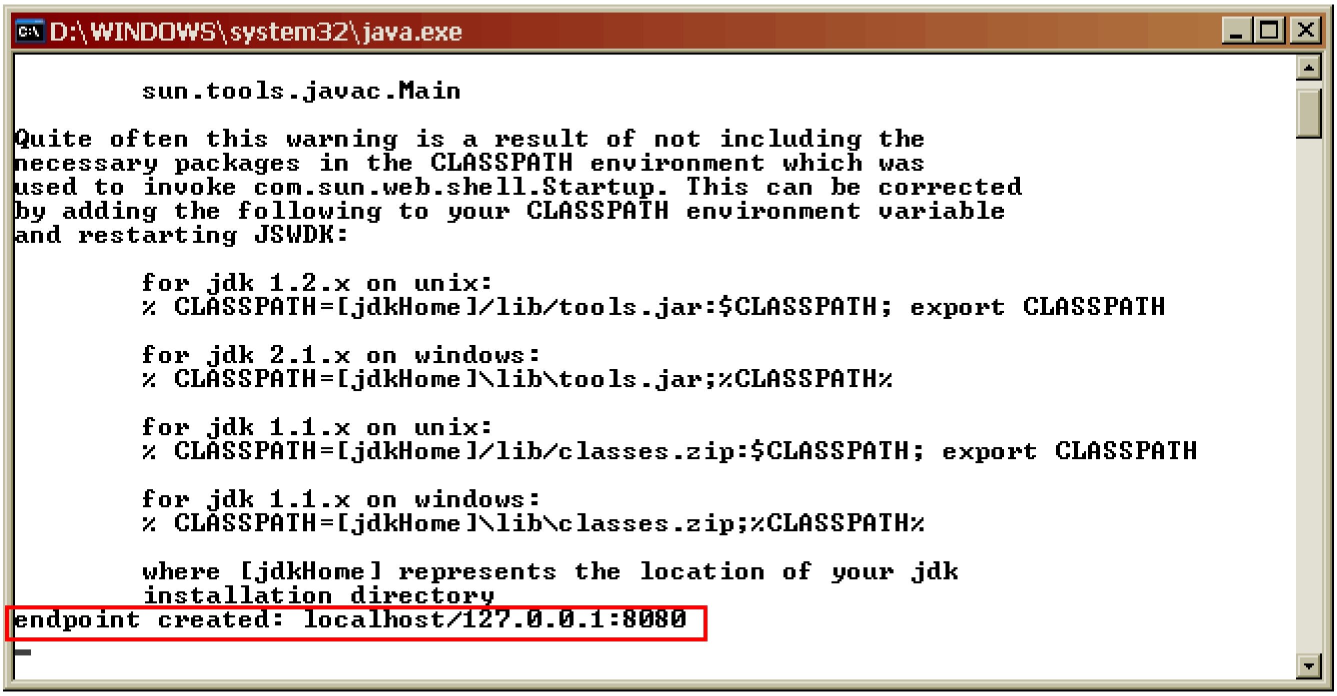
Yaklaşık olarak 745 kb' lık dosyanın windows versiyonunu bilgisayarıma indirdikten sonra, tek yapmam gereken zip dosyasını D sürücümün altına açmak oldu.



Artık servlet' leri çalıştırabileceğim java web sunucum hazırdı. Lakin yapmam gereken bir takım ayarlar olduğunu öğrendim. Bunlardan ilki CLASSPATH ayarlamasıydı. CLASSPATH' e henüz neden eklendiğini anlmayamadığım bir jar paketinin yolunu bildirmem gerekiyordu. Bu, Java geliştirme kitinin kurulduğu dizindeki, lib klasörü içindeki tools.jar paketi idi.



Java web sunucusunun, barındırdığı servletler ve web sayfalarını işleyebilmesi için öncelikle çalıştırılması gerektiğini, IIS (Internet Informatin Server) daki deneyimlerimden biliyordum. Sisteme kurduğum java web sunucusunu çalıştırabilmek JSWDK' nın kurulduğu klasördeki ServersStart.bat isimli dosyayı çalıştırmam yeterli olacaktı. Bu amaçla bu batch dosyasını çalıştırdım. Bu işlemin ardından, komut satırında aşağıdaki ekran görüntüsünü elde ettim. Sanıyorumki web sunucusu çalışmaya başlamıştı.

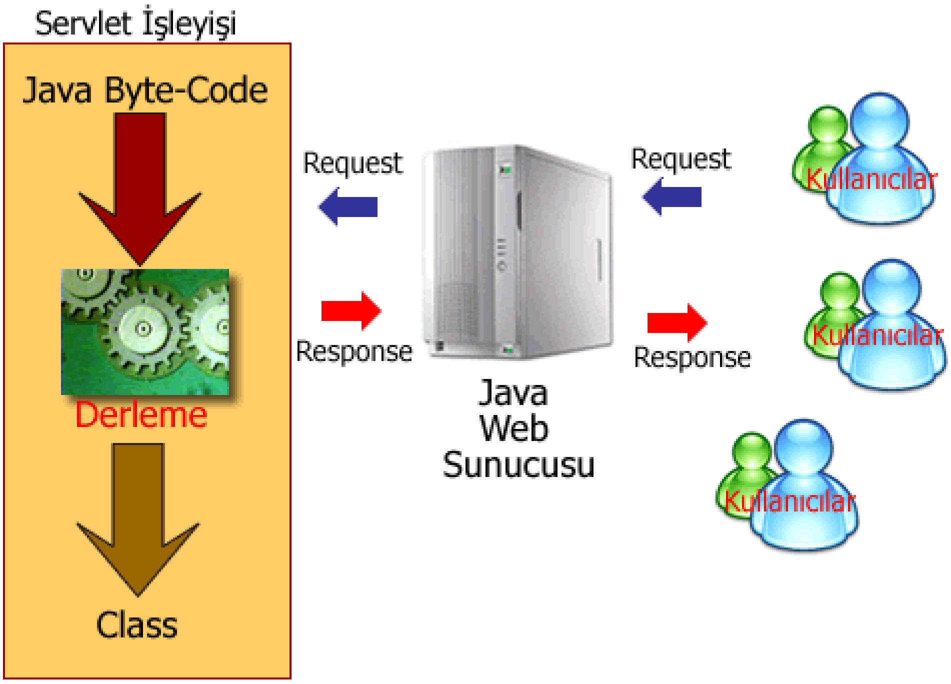


Ancak elbetteki sunucunun çalışıp çalışmadığından bu şekilde emin olmak istemiyordum. Nitekim IIS sistemde çalışırken localhost' u tarayıcı penceresinden talep ettiğinde sistemin çalıştığına ilişkin bir asp sayfası elde ediyordum. Biliyordumki benzer bir strateji java web sunucusu içinde uygulanmalıydı. Bu nedenle kaynaklarımı araştırmaya başladım ve tarayıcı penceresinde, http://localhost:8080/ adresini girdim. Aynen komut satırında bana söylendiği gibi. Sonuç tam istediğim gibiydi.



Artık java web sunucusunun çalıştığını biliyordum. Peki bunu nasıl durduracaktım. StartServer.bat dosyasının çalışması sonucu açılan komut penceresini kapattım. Daha sonra ise, tarayıcı penceresini kapatıp tekrar açtım ve http://localhost:8080/ sayfasını yeniden çağırdım. Ancak sayfa tarayıcıya gelmedi. Yani sunucu çalışması durmuştu. Bununla birlikte aynı işlemi StopServer.bat dosyasını çalıştırarakta gerçekleştirebilirdim. Artık, çalışan bir java web sunucum olduğuna göre basit bir servlet uygulaması yazabilirdim. Servlet' ler teknik olarak, java byte-code dosyalarından ibaretti.

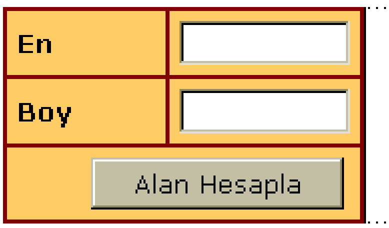
Bu dosyalar, java web sunucusu çalıştırıldığında devreye giren JVM tarafından, sınıf olarak derleniyor ve taleplerin işlenmesi için kullanılan dinamik nesnelere dönüşüyordu. Aslında big picture' ın servlet tarafındaki izahı aşağıdaki şekil ile basitde olsa ifade edilebilirdi.



İlk olarak, Frontpage ile aşağıdaki HTML kodlarından oluşan basit bir form sayfası yaptım. Amacım, kullanıcının bu sayfada girdiği verileri, bir servlet ile almak, işlemek ve bir sonuç sayfası üreterek kullanıcıyla göndermek.

```
<html>
<head>
<title>Alan Hesaplar</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1254">
<meta http-equiv="Content-Language" content="tr">
</head>
<body>
   <form action="http://localhost:8080/servlets/AlanHesap"
method="get">
        <table border="1" cellpadding="5" cellspacing="0"
width="17%" bordercolor="#800000" bgcolor="#FFCC66"
id="table1">
            <tr>
```

```
            <td width="28"><font
size="2"><b>En</b></font></td>
            <td width="44%"><input type="text" name="EN"
size="11"></td>
         </tr>
         <tr>
            <td width="28"><font
size="2"><b>Boy</b></font></td>
            <td width="44%"><input type="text" name="BOY"
size="11"></td>
         </tr>
         <tr>
            <td colspan="2">
            <input type="submit" value="Alan Hesapla"
name="B1" style="float: right"></td>
         </tr>
      </table>
   </form>
</body>
</html>
```



Oluşturduğum bu basit AlanHesaplar.html isimli sayfayı, java web sunucusunda çalıştırabilmek için, D:\jswdk-1.0.1\webpages klasörü altına kaydettim. Bu html sayfasındaki belkide en önemli nokta, Form' daki submit butonuna basılması halinde, form bilgilerinin hangi adreste işleneceğidir. Bunun için,

```
<form method="get"
action="http://localhost:8080/servlets/AlanHesap">
```

satırını kullandım. Burada açıkça görüldüğü gibi, java web sunucusu üzerindeki AlanHesap isimli bir servlet, bu formdaki verileri işleyecekti. Servlet' ler, java web sunucusunda genellikle, D:\jswdk-1.0.1\webpages\WEB-INF\servlets

adresinde tutulmaktaydı. Bende bu kuralı değiştirmedim. Şimdi, web sayfamdaki içeriği işleyecek ve talepleri karşılayacak, AlanHesap isimli servlet' ime ait java byte-code dosyasını oluşturmam gerekiyordu. Bunun için, yine notepad editörü ile, D:\jswdk-1.0.1\webpages\WEB-INF\servlets klasörü altında, aşağıdaki kodlara sahip AlanHesap.java dosyasını oluşturdum.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.*;

public class AlanHesap extends HttpServlet
{
    public void doGet(HttpServletRequest talep,
HttpServletResponse cevap) throws IOException,
ServletException
    {
        cevap.setContentType("text/html");

        PrintWriter cikti=cevap.getWriter();

        String en=talep.getParameter("EN");
        String boy=talep.getParameter("BOY");

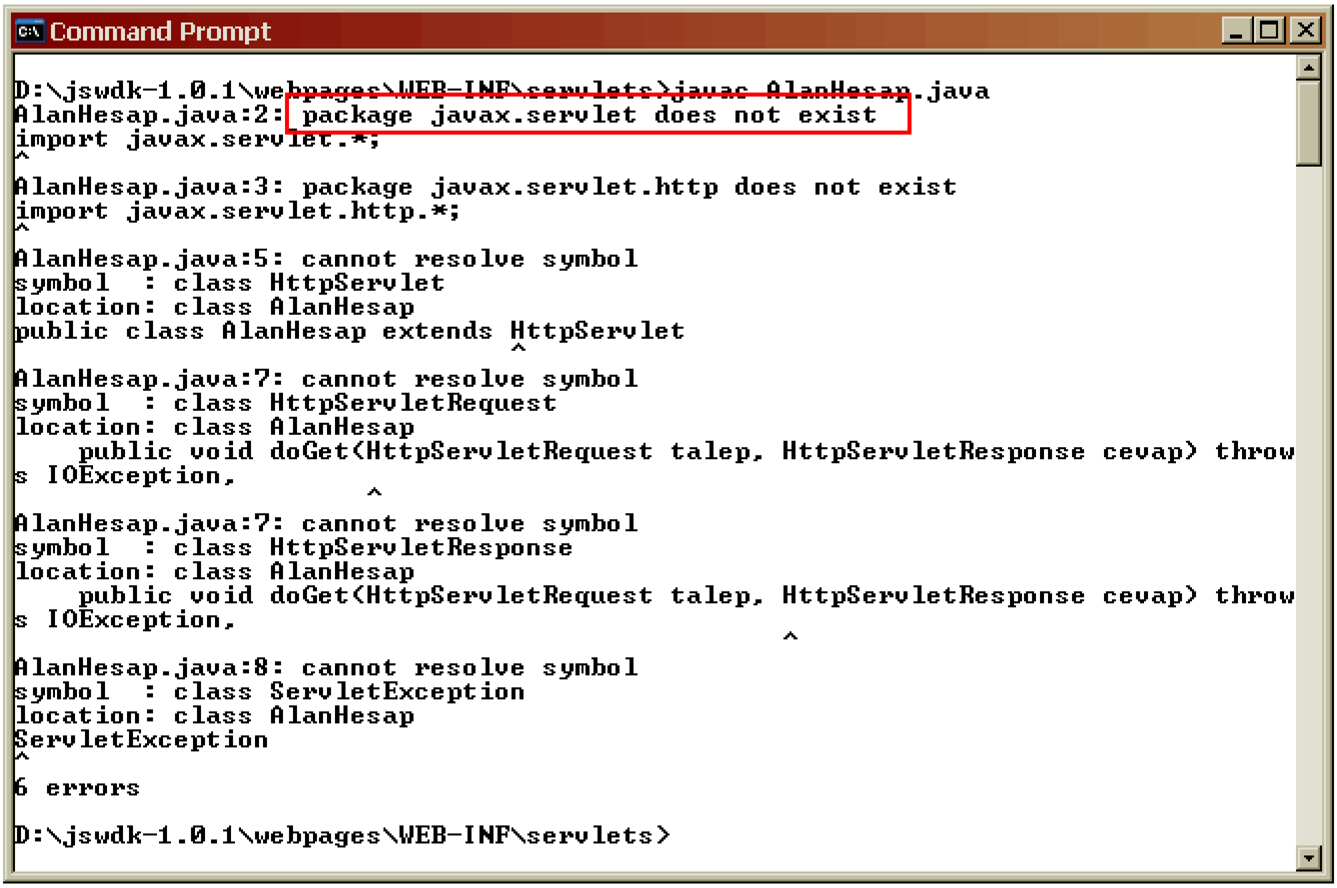
        double eni=java.lang.Double.parseDouble(en);
        double boyu=java.lang.Double.parseDouble(boy);

        double alan=eni * boyu;

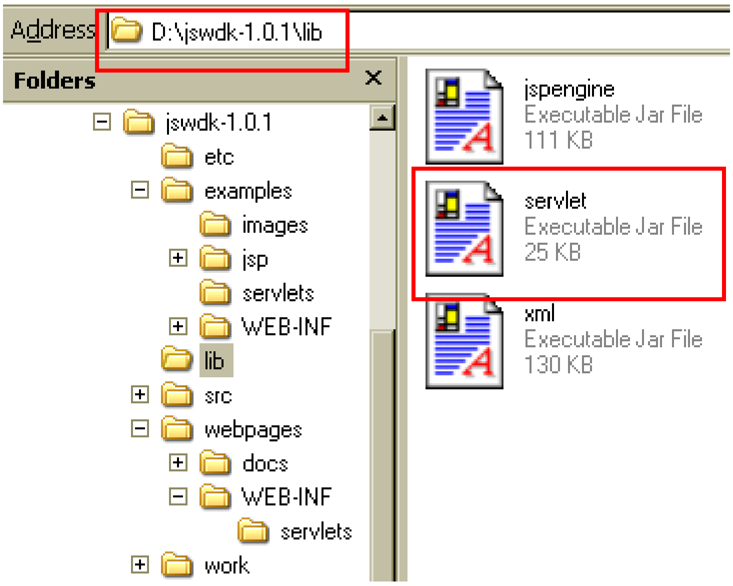
        cikti.println("<b>"+en+" x "+boy+" = "+alan+"</b>");

    }
}
```

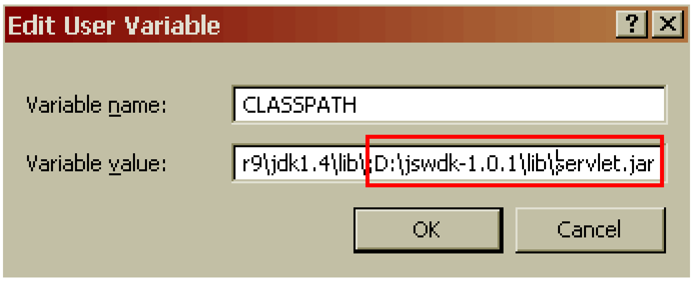
Hemen kod satırından derleme işlemine geçtim. Ancak oda ne? 6 tane hata mesajı aldım. Tam anlamıyla yıkılmıştım.



Ancak hata koldarına baktığımda ve javax.servlet isimli paketin bulunamadığını anladığımda, ne yapmam gerektiğini biliyordum. Hemen, servlet isimli bir paket aramaya başladım. Neyseki, D:\jswdk-1.0.1\lib klasöründe bir tane vardı. Bu paket içinde, kodlarda kullandığım HttpServletResponse, HttpServletRequest, HttpServlet gibi sınıflar yer almaktaydı.



Hemen bu paketi CLASSPATH tanımlamalarımın sonuna aşağıdaki gibi ekledim.



Byte-Code dosyasını yeniden derlediğimde hatasız bir şekilde derleme işleminin gerçekleştirildiğini gördüm. Sayfamı çalıştırmak ve servlet' in işleyip işlemediğini görmek için sabırsızlanıyordum. Lakin öncesinde neler yaptığımı anlamam gerektiğine karar verdim ve kod satırlarını incelemeye başladım. Herşeyden önce yazdığım uygulama bir servlet olacağı için, HttpServlet sınıfından türetilmişti. doGet metodunun ismi rastgele verilmiş bir isim değildi. Form üzerindeki bilgileri servlet' e geçirirken metod olarak get tekniğini kullanacağımı belirtmiştim. Bu nedenle get tekniği ile çalışacak form sayfaları için doGet metodu kullanılıyordu. Eğer form üzerindeki bilgileri post tekniğine göre göndermek isteseydim bu durumda doPost metod ismini kullanmam gerekecekti. Elbette form üzerindeki method takısınıda post olarak ayarlamam gerekirdi. doGet metodu içinde iki önemli parametre yer almaktaydı.

```
public void doGet(HttpServletRequest talep,
HttpServletResponse cevap) throws IOException,
ServletException
```

HttpServletRequest parametresi ile, kullanıcların tarayıcalarından gelen talepleri alabiliyordum. HttpServletResponse ilede, tarayıcılara cevap gönderebiliyordum. HttpServletRequest nesnesinin getParameter metodu ile, form üzerindeki EN ve BOY metin kutularına ait değerleri alabilmekteydim.

```
String en=talep.getParameter("EN");
String boy=talep.getParameter("BOY");
```

Çıktı üretmek için, IO paketinde yer alan, PrintWriter sınıfını kullanıyordum. Bu sınıfa ait nesneyi oluştururken, HttpServletResponse nesnesinin getWriter metodunu kullanmaktaydım.

```
PrintWriter cikti=cevap.getWriter();
```

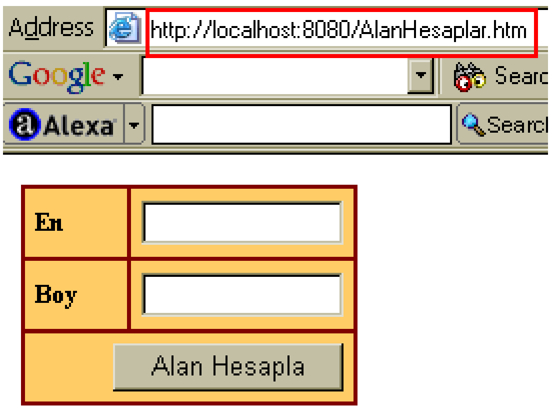
Böylece, PrintWriter nesnesinin örneğin println metodu ile, talepte bulunan tarayıcıya bir şeyler yazdırabilirdim.

```
cikti.println("<b>"+en+" x "+boy+" = "+alan+"</b>");
```

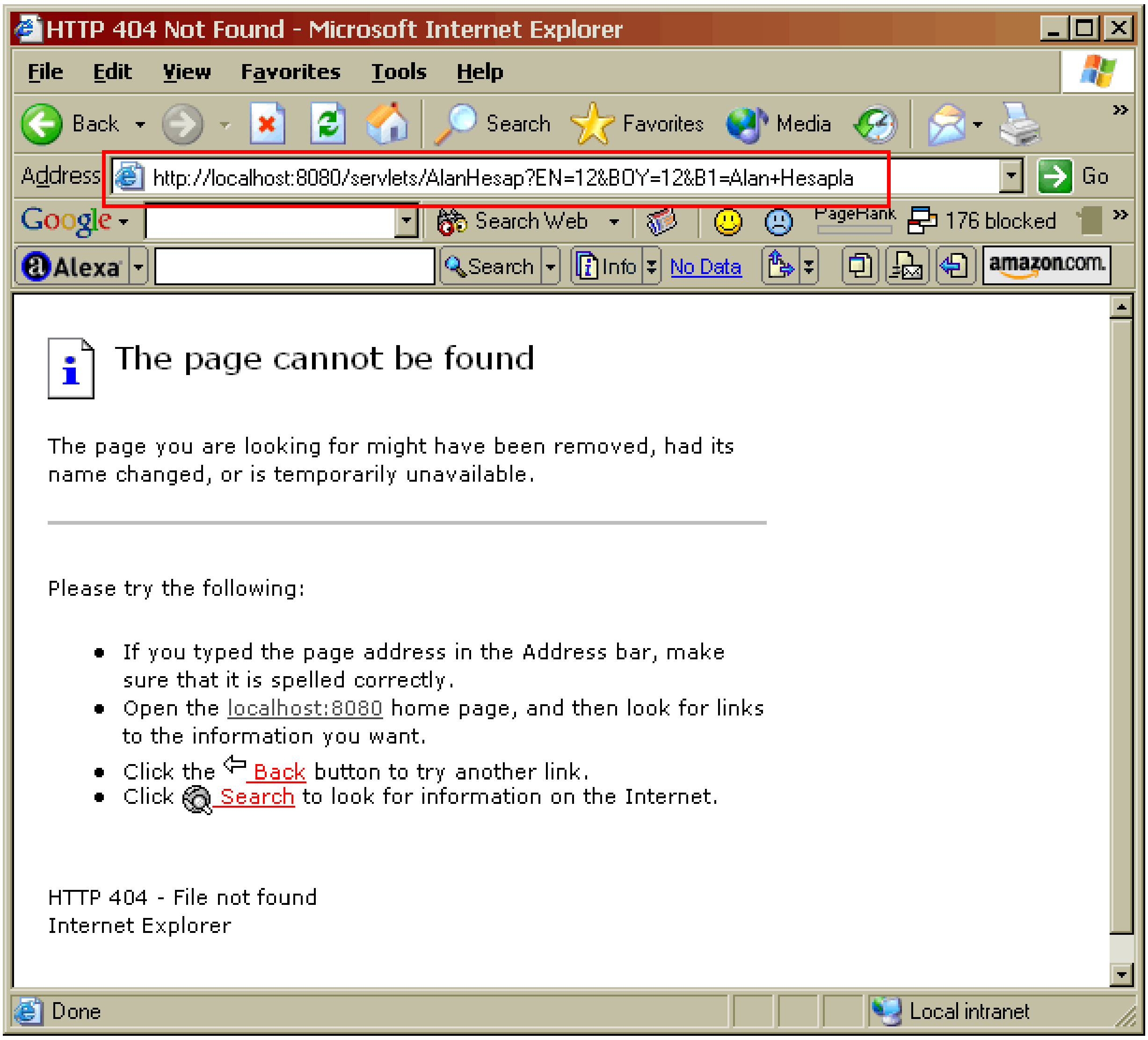
Az çok, servlet' ler ile dinamik kodlamanın nasıl gerçekleştirilebileceğini anlamıştım. Elbetteki, servlet' ler ile ilgili konuların daha derin olduğunu ve öğrenmem gereken daha pek çok şey olduğunu biliyordum. Bu düşünceler içerisindeyken, sayfamı test etmeye başladım. Sonuçta pek bir işe yaramayan bir çalışma olacağı kesindi. Ancak en azından bir tarayıcıdaki kullanıcı hareketlerini, sunucu taraflı kodlama tekniği ile nasıl ele alabileceğimi keşfetmiştim. Hemen sayfamı denemeye karar verdim ve tarayıcı pencereme aşağıdaki linki girdim.

```
http://localhost:8080/AlanHesaplar.htm
```

tabiki sayfa çalışmadı. Çünkü, java web sunucusunu henüz çalıştırmamıştım. Hemen, StartServer.bat dosyası ile sunucuyu çalıştırdım ve tekrar sayfamı yükledim. Bu kez sayfam çalışmıştı.



Evet, şimdi forma bir şeyler yazıp servlet' imin çalışmasını seyredebilirdim. İşte sonuç. Gümmmmmmm...



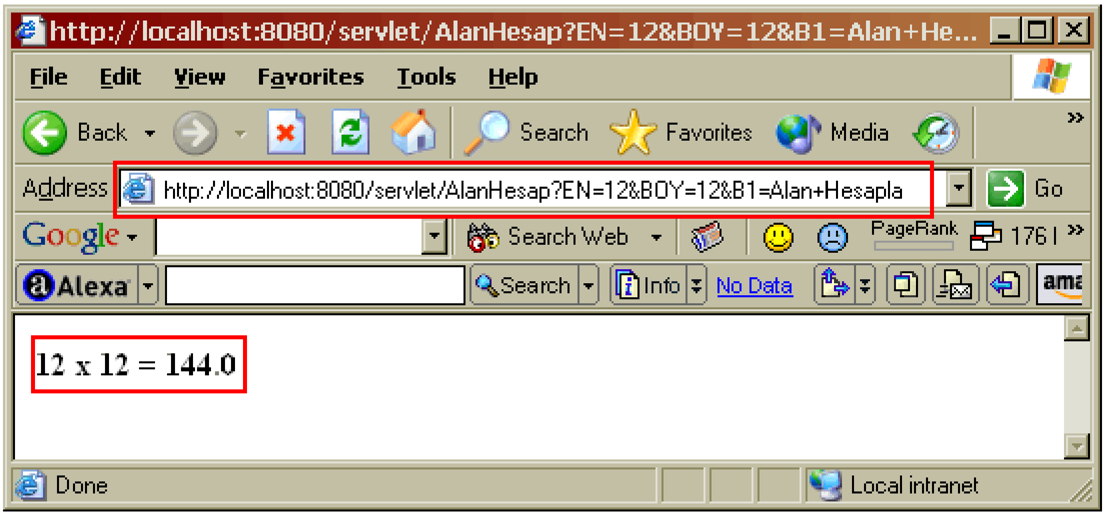
Ne olmuştuda, servlet bulunamamış ve çalıştırılmamıştı. Herşeyi tekrar tekrar gözden geçirdim. Kodlarda bir hata yoktu. Aksi halde uygulama derlenmezdi. Sonra servlet dosyalarını yanlış bir klasöre koymuş olabileceğimi düşündüm. Ancak bu da değildi. Bu arada kendime bir kahve daha yaptım ve evin bir ucundan diğerine yürümeye başladım. Bir yudum alıyor kendi kendime "alla allaaa" diyordum. Sonra tabiki dayanamayıp kaynaklarıma bakmaya karar verdim. Meğerse sorun s harfindeymiş. Eğer bu proje bir uzay programı olsaydı sanırım mekik bu s harfi yüzünden çoktan aya çarpmış olurdu. Adres satırındaki,

```
http://localhost:8080/servlets/AlanHesap?EN=Java24&BOY=Java24&B1=G
```

ifadesini,

```
http://localhost:8080/servlet/alanHesap?EN=Java24&BOY=Java24&B1=G
```

satırı ile değiştirdiğimde sorun çözülmüştü. Aradaki fark gerçektende tek bir s harfiydi. Servlet' imi servlets isimli klasör altına atmıştım. Ancak sanal suncuda yazılan adreste servlets yerine servlet kullanmam gerekiyordu. Hemen giris.htm sayfasındaki kodu düzelttim. Bu kez başarmıştım. Servlet çalışmış, form üzerindeki bilgiler işlenmiş ve sonuç üretilmişti.



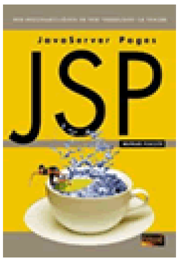
Artık servlet' leri az çokda olsa anlamıştım. Sonuçta, web sayfalarının sunucu taraflı kısımlarında dinamik etkileşime izin veren kodlamalar java dili ile gerçekleştirildiğinden bundan sonra işim daha da kolaydı. Lakin kahvemin son yudumlarını içerken, JSP olayının ne olduğunu düşünmeye başladım. Sanırım önümüzdeki hafta JSP olayına gireceğim.

Bölüm 24: İlk Bakışta JSP (Java Server Pages)

Bu hafta, JSP ile dinamik etkileşimli bir web sayfası oluşturmak için kendimi bayağı bir hırpaladım. Sorunlar, JSP olarak yazdığım örnek sayfaları, servlet' leri çalıştırmak için kullandığım JSWDK kitinde baş gösterdi. Nedense hiç bir JSP

sayfasını çalıştıramadım. Evin içinde bir o köşeye bir bu köşeye koşuştururken nerde hata yaptığımı bulmaya çalışıyordum. Sorun büyük bir ihtimalle benim sistemimden kaynaklanıyordu. Tabiki benimde yaptığım yanlış ayarlamalar olabilirdi. Sonuda kurcalaya kurcalaya güzelim JSWDK kitinide bozuvermiştim.

Ne yapabilirim diye düşünürken, aklıma parlak bir fikir geldi. Elimdeki kaynaklar JSP uygulamalarına ufak bir dokunuş yapmıştı. Ancak bu konuda esaslı bir kaynağım olması gerektiği inancındaydım. Her ne kadar amacım mimarinin temellerini ve çalışma sistemini anlayıp çok basit bir uygulama geliştirmek olsada, yinede elimin altında her zaman başvurabileceğim bir kaynak olması gerektiğini düşünüyordum. Bu düşünceler ile hemen yola koyuldum ve Kadıköy' deki kitapçıları gezmeye başladım. Sonunda aradığım kitabı bulmuştum. Pusula yayınlarından Numan Pekgöz' ün JSP isimli kitabı.



İşin güzel yanı, kitapta JSP sayfalarını kurup çalıştırabileceğim uygulamaların yer aldığı bir CD' de mevcuttu. Büyük bir heyecan ile kiabı aldım ve yol boyunca otobüste okuyarak geldim. Eve geldiğimde, JSP sayfalarını çalıştırabileceğim JRun sunucusunun nasıl kurulacağını ve kullanılacağını çoktan öğrenmiştim bile. Hemen kitapta bahsedilen adımları, birer birer işlemeye başladım. JRun, şu anda Macromedia firmasına ait olan bir ürün olmakla birlikte kitabın bu sürümünde bir önceki verisyonuna yer verilmiş. Ama hiç önemli değil. Sonuçta JSP sayfaları çalışıyor. Kurulumu bitirdiğimde,

```
D:\Program Files\Allaire\JRun\
```

dizinine, java web sunucum kurulmuş, hazır ve nazırdı. Artık geliştirmek istediğim JSP sayfalarını

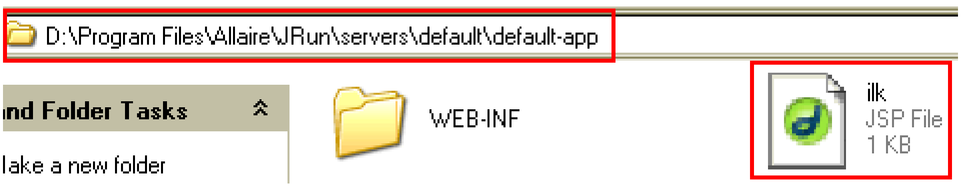
```
D:\Program Files\Allaire\JRun\servers\default\default-app\
```

klasörü altına atacaktım. Tabi kurulum basit olmasına rağmen yinede sistemde yapılması gereken ince ayarlamalarda vardı. Bunlarda birisi her zamanki gibi meşhur ClassPath tanımlamaları ile ilgiliydi. JSP sayfalarının başarılı bir şekilde işlenebilmesini sağlamak için,

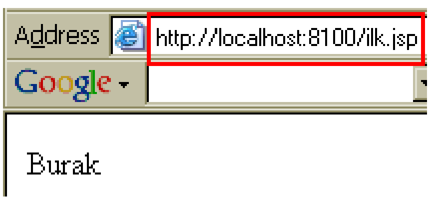
```
D:\Program Files\Allaire\JRun\lib\ext\servlet.jar
```

adresindeki servlet.jar paketinin Classpath tanımına eklenmesi gerekiyordu. Bu işlemin ardından hemen ilk olarak aşağıdaki gibi basit bir sayfa oluşturdum ve bu sayfayı jsp uzantısı ile, D:\Program Files\Allaire\JRun\servers\default\default-app klasörü altına kaydettim.

```
<html>
<head>
<title>ilk jsp</title>
<body>
<%
String adim="Burak";
out.println(adim);
%>
</body>
</html>
```



Şimdi tek yapmam gereken, tarayıcı penceresinde, http://localhost:8100/ilk.jsp adres satırını girmekti. Bende öyle yaptım.

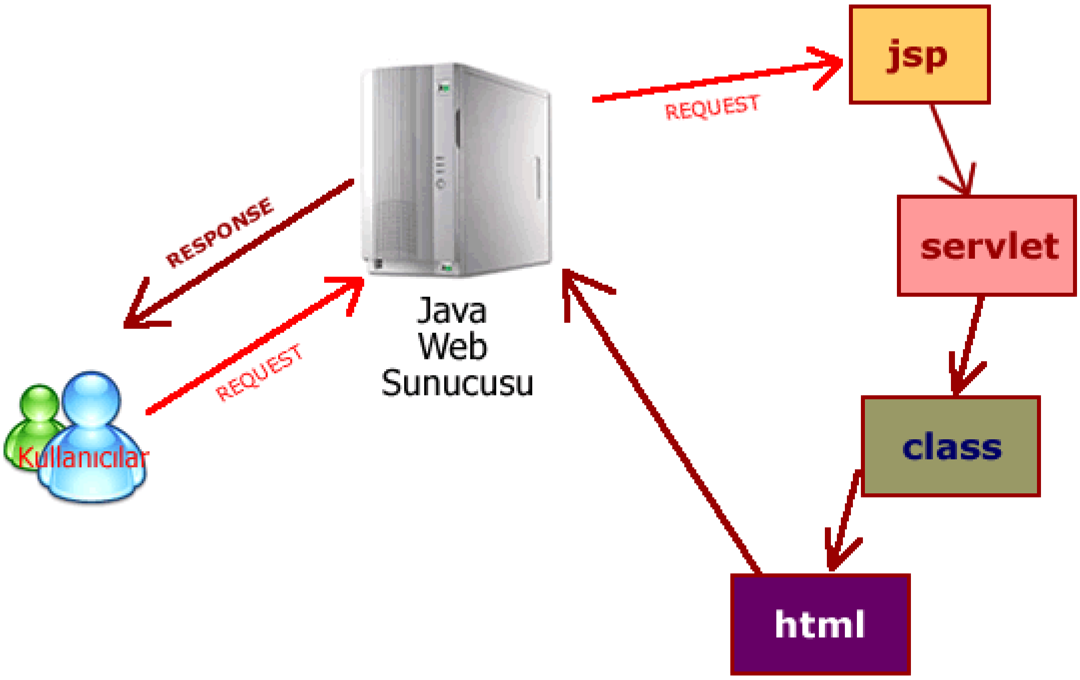


Şaka maka, ilk JSP sayfamı yazmıştım. Evet herşey iyidi hoştu ama, bu JSP nedemekti? Ne işe yarıyordu? Herşeyden önemlisi

çalışma sistemi neydi? JSP ile ilgili tüm kaynaklarımdaki bilgilerimi şöyle bir gözden geçirdim ve büyük resme(big picture) ulaşmak için adımlarımı atmaya başladım.

Herşeyden önce, geçen hafta servlet' ler yardımıyla, kullanıcı ile dinamik etkileşimli web sayfalarının nasıl gerçekleştirilebileceğini incelemiştim. JSP (Java Server Pages)' lerde aynı işi yapmaktaydı. Aralarındaki farklılıkları düşündüğümde ilk akla gelen, JSP'lerde html içerik ile Java dilinin kullanıldığı sunucu taraflı (server-side) kodların bir arada bulunmasıydı. Öyleki servlet'ler ile çalışırken, java kodlarını kullanarak html çıktılarını üretmek gerçekten zahmetli bir işti. Açıkçası hammallıktı. Oysaki JSP'de html içerik ile java kodları aynı sayfada yer alabilmekteydi. Bu ise, tasarımın etkili olduğu, sunucu taraflı java kodları barındıran web sayfalarının kolayca oluşturulabilmesi demekti.

Bu fark önemsiz gibi görünsede, geliştirme zamanında oldukça büyük tasarruf sağlamakta. Yaklaşık olarak 8 saatlik bir JSP' ci olarak bunu ben bile söyleyebiliyorum. Diğer yandan bence asıl önemli olan fark, JSP sayfalarının işleyiş şekli. Öyleki, oldukça ilginç bir durum söz konusu. O da, JSP' lerin, istemciler tarafından talep edildiklerinde, eğer bu talep ilk kez gerçekleşmişse, java kodlarının, servlet'ler haline getirilmesi ve sınıf olarak derlenmesi. Yani, eninde sonunda yazılan JSP' ler derlencekleri zaman işin içine servlet' ler girmekte. Kısacası JSP'ler bizi servlet yazma derdinden kurtarıyor diyebiliriz. Ancak tabiki Jsp' lerin servlet' ler ile tümleşik olarak çalışmasıda söz konusu. Aslında Java sunucu sayfalarının çalışma prensibini aşağıdaki şekil ile daha iyi anlatabileceğime inanıyorum.

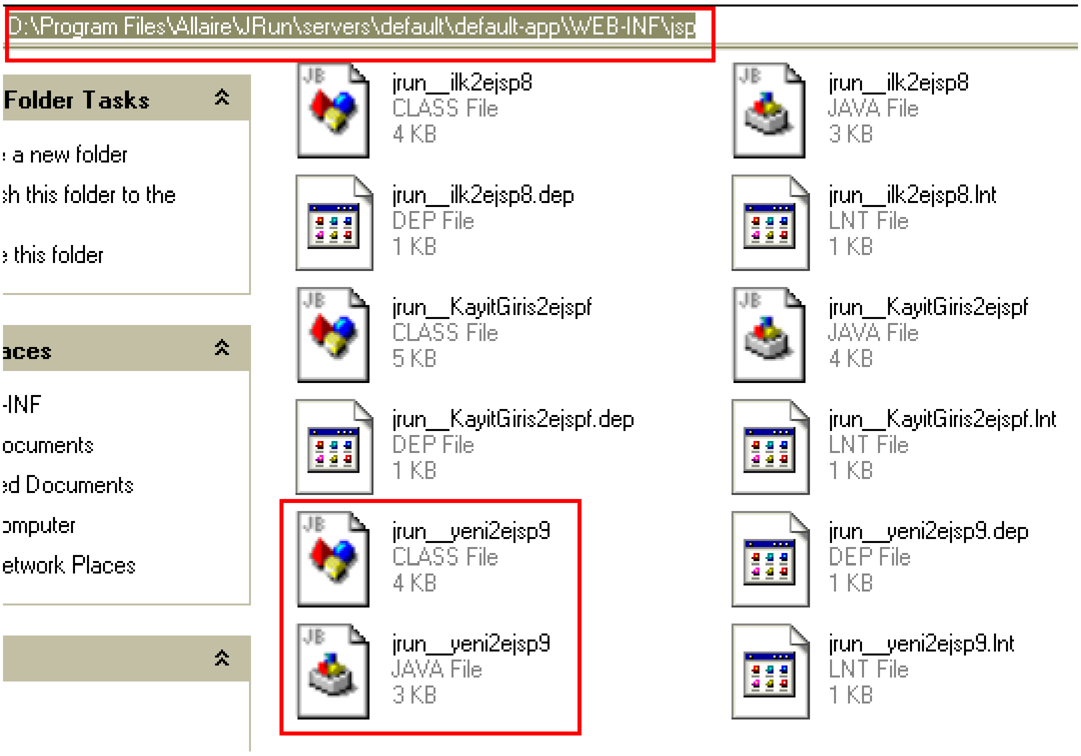


Jsp sayfalarının çalışma prensibi bu şekilde. Yani, kullanıcı bir JSP sayfasını ilk kez talep ettiğinde, sunucu bu talep üzerine ilgili JSP sayfasının java kodlarından bir servlet oluşturuyor. Bu servlet dosyası aslında bir java byte-code dosyası. Sonraki adımda ise byte-code dosyası class olarak derleniyor. Derlenen class, kullanıcıdan gelen parametreler vs... varsa bunlarla birlikte çalıştırılıyor ve üretilen sonuçlar html olarak tekrardan java sunucusuna gönderiliyor. Java sunucusu ise, bu html sonuçlarını kullanıcının tarayıcısına gönderiyor. Bu sistem, sadece oluşturulan Jsp sayfası kullanıcı tarafından ilk kez talep edildiğinde gerçekleştiriliyor.

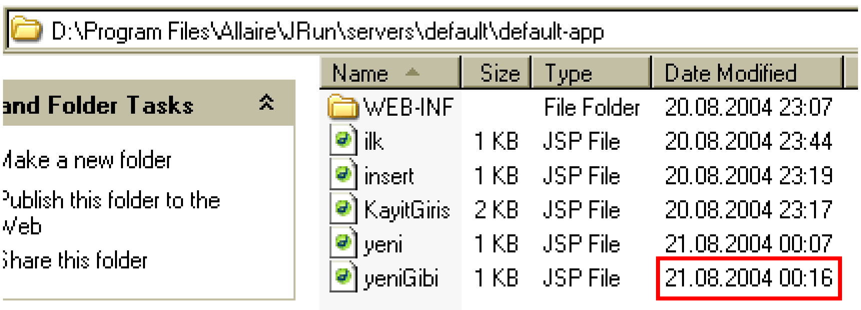
Nitekim bundan sonraki çağrılarda zaten var olan derlenmiş class dosyları çalışıyor. Elbette biz Jsp sayfamızı değiştirirsek buradaki süreç tekrarlanacaktır. Peki bu teori gerçekten böylemi? Bunu bir şekilde ispat etmem gerektiğini düşündüm kendimce. Geliştirdiğim, ilk.jsp dosyasının aynısının farklı isimli bir verisyonunu oluşturdum. Yeni.jsp olarak. Daha sonra bu sayfayı tarayıcıdan talep ettim. Sonra, alt klasörleri gezinmeye başladım. Nemi arıyordum? İçinde Yeni kelimesi olan java ve class uzantılı dosyalar. Eğer bu dosyaları bulursam gerçektende teorinin çalışmasını ispatlamış olucaktım. Derken,

D:\Program Files\Allaire\JRun\servers\default\default-app\WEB-INF\jsp

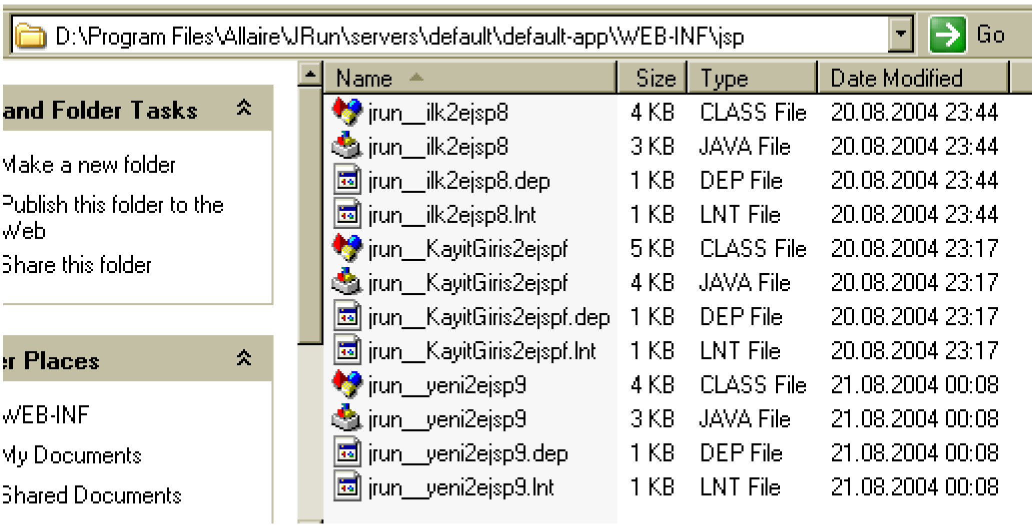
klasöründe, bu dosyaları buldum.



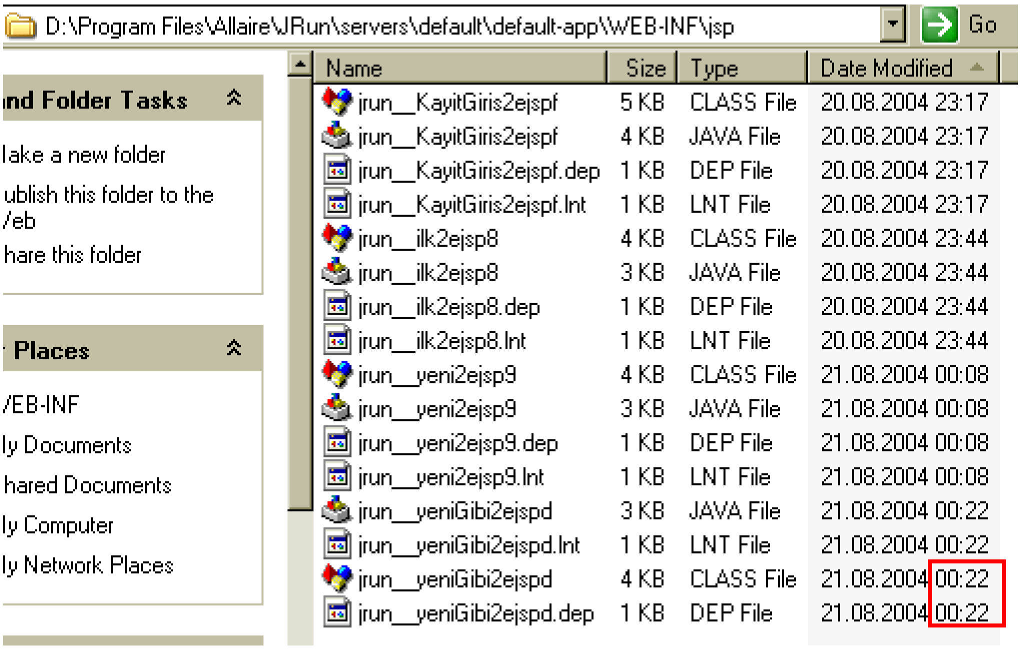
Ancak bu ispatta eksik olan bir şeyler var gibiydi. Belkide ben Jsp dosyasını kaydettiğimde, bu dosyalar otomatik olarak oluşturulmuştu. İşte bu hipotezi yıkmanın tek bir yolu vardı. Yeni bir jsp dosyası oluşturacak, bu sayfanın oluşturuluş zamanını kaydedecek sonra bir kahve molası verecektim. Geldiğimde tarayıcıdan bu sayfayı çalıştıracak ve oluşan java ve class dosyalarının süresini, jsp dosyasının oluşuturuluş süresi ile karşılaştıracaktım. Bunun üzerine, Yeni.Jsp' den bir kopya daha oluşturdum. YeniGibi.jsp.



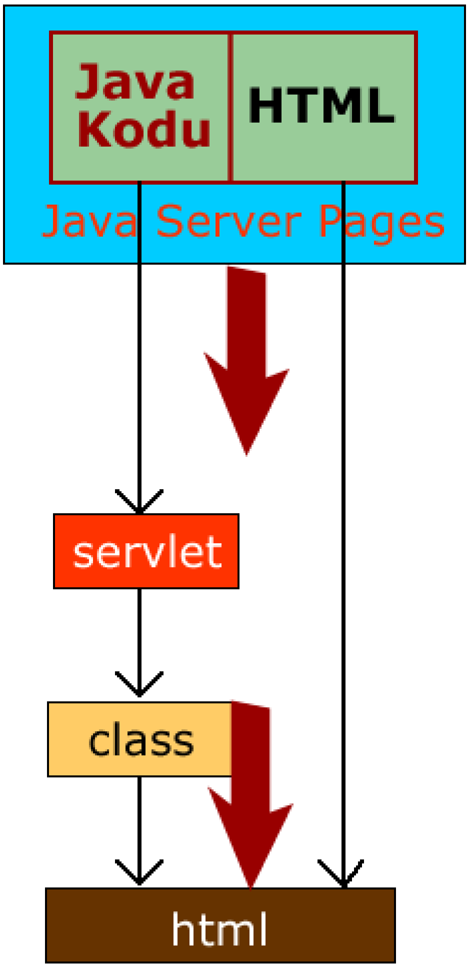
Şimdi de kahve molasına çıkma zamanı. Neyse, kahve molasına çıkıp geldim. Bir kaç dakika sonra geldiğimde hemen, jsp klasörünün içine baktım.



Görünürde YeniGibi.jsp için oluşturulmuş java ve class dosyalarından eser yoktu. Derken tarayıcıda bu sayfayı tekrardan çalıştırdım ve jsp klasörüne tekrar baktım.



Artık ispat gerçekleşmişti. Jsp ler ile ilgili öenmli bir nokta, html sayfaları ile içiçe yazılabilmeleriydi. Aynı asp sayfalarında olduğu gibi. Tabi java ile yazılan bu sunucu taraflı kodlar, tam anlamıyla nesne yönelimli bir dilin avantajlarını kullandığından oldukça esnek ve güçlü sonuçlar elde etmemizi sağlamaktaydı. Peki içiçe çalışan bu sayfaların işleyişine yakından bakmak isteseydim nasıl bir şekli kafamda canlandırabilirdim diye düşünmeye başladım. Sonuçta aşağıdaki nacizane şema ortaya çıktı.



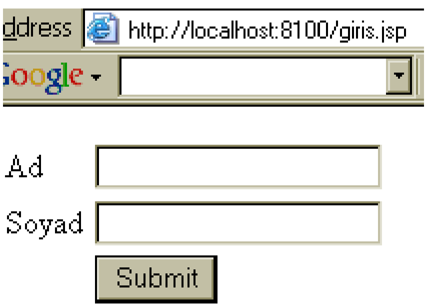
Bu şema aslında, Jsp' nin bir avantajınıda göstermekte. O da, tasarım ve kodlama katmanlarının ayrı ayrı ele alınması. Yani, bir tasarımcı ve bir java programcısı kafa kafaya verdiklerinde, dünyanın en çok konuşulan etkileşimli web sayfalarını yazabilirler. Kodlamacı java dili ile sayfayı kodlarken, tasarımcı sadece görsel tasarım ile ilgilenecektir. Gerçi ben, Asp.Net ile gelen code-behind modelini bu modele tercih ederim. Nitekim oradada kod katmanı ve tasarım katmanı ayrıdır. Hatta o kadar ayrı olabilirlerki, code-behind tekniği sayesinde, aspx sayfalarına ati kodlar C# gibi nesne yönelimli bir dil ile ayrı sınıf dosyalarında toparlanabilir. Tabiki .net içindeki bu teknik servletlere nazaran, kod ve sunum katmanının daha iyi entegre çalışmasını sağlayan bir yapı üzerinde teşkil edilmiş.

Bu kadar laf kalabalığından sonra aslında işe yarar bir örnek yapmanın zamanı geldi. İşe yarar dediysem ufkumu açmak için teşebbüste bulunacak işe yaramaz bir uygulamdan bahsediyorum elbette. En azından, bir jsp sayfasındaki form bilgilerinin başka bir jsp sayfası tarafından ele alınmasını

sağlamak gibi basit bir işlem gerçekleştirebilirim. Bu amaçla aşağıdaki kodları içeren jsp sayfalarını oluşturdum.

Giris.jsp sayfası;

```
<html>
<head>
<title>ilk jsp</title>
<body>
<form name="form1" method="post" action="kontrol.jsp">
<table width="200" border="0">
<tr>
<td>Ad</td>
<td><input type="text" name="tAd"></td>
</tr>
<tr>
<td>Soyad</td>
<td><input type="text" name="tSoyad"></td>
</tr>
<tr>
<td></td>
<td><input type="submit" name="Submit"
value="Submit"></td>
</tr>
</table>
</form>
</body>
</html>
```



kontrol.jsp

```
<html>
<head>
```

```
<title>Giriş</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
</head>

<body>
<table width="200" border="0">
<tr>
<td><strong><%=request.getParameter("tAd")%></strong></td>
</tr>
<tr>
<td><strong><%=request.getParameter("tSoyad")%></strong></td>
</tr>
</table>

</body>
</html>
```

Yazdığım kodlarda tek göze çarpan nokta, form' daki field' lara ait değerleri, kontrol.jsp dosyasında, request sınıfının getParameter metodu ile alıyor oluşumdu. Hemen, giriş.jsp sayfasını sunucuda çalıştırdım ve bilgileri girip buton kontrolüne bastım. Sonuçta, kontrol.jsp sayfam çalıştı ve aşağıdaki ekran görüntüsünü elde ettim.



Jsp oldukça geniş bir konu ve incelenmeye değer. Şu an için çalışma prensibini, en büyük avantajını ve nasıl geliştirildiğini biliyorum. Ancak elbetteki bir kaç günlük çalışma ile Jsp' ye hakim olamam. Hiç olmassa ilk bakışta göze çarpan noktaları irdelemiş oldum.

Bu kahve molamız ile birlikte Java ile 24 Kahve Molasının' da sonuna gelmiş bulunuyoruz. Ancak bunun bir son olmadığını hatta bir başlangıç olduğunu söylemek isterim. Nitekim

değinemediğim pek çok konu var. Örneğin değerli bir okurumuzun belirttiği gibi ağ programcılığı. Bunun dışında, thread' ler, koleksiyonlar (torbalar), IO (Input-Output) işlemleri vb... Dolayısıyla bu konularada değinmek ve incelemek düşüncesindeyim. Bundan sonrasında, her cuma olmasa bile düzenli olarak bahsettiğim konuları inceleyecek ve yazmaya çalışacağım.

Umuyorum ki bu yazı dizisinde sizlere Java dilini bir nebze olsun öğretebilmişimdir. Ben bir şeyler öğrendim. Ama elbetteki öğreniklerimi tekrar etmessem, gerçek projelerde kullanmassam kolayca unutabilirim. Sizlere de tavsiyem bol bol tekrar etmeniz ve projeler geliştirmeniz olacaktır. Hepinize mutlu ve sağlıklı günler dilerim. Salıcakla kalın.

Kaynak: http://www.bsenyurt.com/